

PowerPoint add-ins documentation

With PowerPoint add-ins, you can use familiar web technologies such as HTML, CSS, and JavaScript to build a solution that can run in PowerPoint across multiple platforms, including on the web, Windows, Mac, and iPad. Learn how to build, test, debug, and publish PowerPoint add-ins.

About PowerPoint add-ins

OVERVIEW

[What are PowerPoint add-ins?](#)

[JavaScript API for PowerPoint](#)

QUICKSTART

[Build your first PowerPoint add-in](#)

[Explore Office JavaScript API using Script Lab](#)

HOW-TO GUIDE

[Test and debug a PowerPoint add-ins](#)

[Deploy and publish a PowerPoint add-ins](#)

Key Office Add-ins concepts

OVERVIEW

[Office Add-ins platform overview](#)

GET STARTED

[Core concepts for Office Add-ins](#)


[Design Office Add-ins](#)

[Develop Office Add-ins](#)

Resources

REFERENCE

[Ask questions](#) 

[Request features](#) 

[Report issues](#) 

[Office Add-ins additional resources](#)

PowerPoint add-ins

06/13/2025

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

- Use **task pane add-ins** to bring in reference information or insert data into the presentation via a service. For example, see the [Pexels - Free Stock Photos](#) add-in, which you can use to add professional photos to your presentation. To create your own task pane add-in, you can start with [Build your first PowerPoint task pane add-in](#).
- Use **content add-ins** to add dynamic HTML5 content to your presentations. For example, see the [LucidChart Diagrams for PowerPoint](#) add-in, which injects interactive diagrams from LucidChart into your deck. To create your own content add-in, start with [Build your first PowerPoint content add-in](#).

PowerPoint add-in scenarios

The code examples in this article demonstrate some basic tasks that can be useful when developing add-ins for PowerPoint.

Add a new slide then navigate to it

In the following code sample, the `addAndNavigateToNewSlide` function calls the [SlideCollection.add](#) method to add a new slide to the presentation. The function then calls the [Presentation.setSelectedSlides](#) method to navigate to the new slide.

JavaScript

```
async function addAndNavigateToNewSlide() {
  // Adds a new slide then navigates to it.
  await PowerPoint.run(async (context) => {
    const slideCountResult = context.presentation.slides.getCount();
    context.presentation.slides.add();
    await context.sync();

    const newSlide =
context.presentation.slides.getItemAt(slideCountResult.value);
    newSlide.load("id");
    await context.sync();

    console.log(`Added slide - ID: ${newSlide.id}`);
  });
}
```

```

    // Navigate to the new slide.
    context.presentation.setSelectedSlides([newSlide.id]);
    await context.sync();
  });
}

```

Navigate to a particular slide in the presentation

In the following code sample, the `getSelectedSlides` function calls the `Presentation.getSelectedSlides` method to get the selected slides then logs their IDs. The function can then act on the current slide (or first slide from the selection).

JavaScript

```

async function getSelectedSlides() {
  // Gets the ID of the current slide (or selected slides).
  await PowerPoint.run(async (context) => {
    const selectedSlides = context.presentation.getSelectedSlides();
    selectedSlides.load("items/id");
    await context.sync();

    if (selectedSlides.items.length === 0) {
      console.warn("No slides were selected.");
      return;
    }

    console.log("IDs of selected slides:");
    selectedSlides.items.forEach(item => {
      console.log(item.id);
    });

    // Navigate to first selected slide.
    const currentSlide = selectedSlides.items[0];
    console.log(`Navigating to slide with ID ${currentSlide.id} ...`);
    context.presentation.setSelectedSlides([currentSlide.id]);

    // Perform actions on current slide...
  });
}

```

Navigate between slides in the presentation

In the following code sample, the `goToSlideByIndex` function calls the `Presentation.setSelectedSlides` method to navigate to the first slide in the presentation, which has the index 0. The maximum slide index you can navigate to in this sample is `slideCountResult.value - 1`.

```

async function goToSlideByIndex() {
  await PowerPoint.run(async (context) => {
    // Gets the number of slides in the presentation.
    const slideCountResult = context.presentation.slides.getCount();
    await context.sync();

    if (slideCountResult.value === 0) {
      console.warn("There are no slides.");
      return;
    }

    const slide = context.presentation.slides.getItemAt(0); // First slide
    //const slide = context.presentation.slides.getItemAt(slideCountResult.value -
1); // Last slide
    slide.load("id");
    await context.sync();

    console.log(`Slide ID: ${slide.id}`);

    // Navigate to the slide.
    context.presentation.setSelectedSlides([slide.id]);
    await context.sync();
  });
}

```

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() {
  // Gets the URL of the current file.
  Office.context.document.getFilePropertiesAsync(function (asyncResult) {
    const fileUrl = asyncResult.value.url;
    if (fileUrl === "") {
      console.warn("The file hasn't been saved yet. Save the file and try
again.");
    } else {
      console.log(`File URL: ${fileUrl}`);
    }
  });
}

```

Create a presentation

Your 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

```
const myFile = document.getElementById("file") as HTMLInputElement;
const reader = new FileReader();

reader.onload = function (event) {
    // Strip off the metadata before the Base64-encoded string.
    const startIndex = reader.result.toString().indexOf("base64,");
    const 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]);
```

To see a full code sample that includes an HTML implementation, see [Create presentation](#).

Detect the presentation's active view and handle the `ActiveViewChanged` event

If you're building a [content add-in](#), you'll need to get the presentation's active view and handle the `Document.ActiveViewChanged` event as part of your `Office.onReady` call.

 **Note**

In PowerPoint on the web, the `Document.ActiveViewChanged` event will never fire because **Slide Show** mode is treated as a new session. In this case, the add-in must fetch the active view on load, as shown in the following code sample.

Note the following about the code sample:

- The `getActiveFileView` function calls the `Document.getActiveViewAsync` method to return whether the presentation's current view is "edit" (any of the view where you can edit slides, such as **Normal**, **Slide Sorter**, or **Outline**) or "read" (**Slide Show** or **Reading View**), represented by the `ActiveView` enum.
- The `registerActiveViewChanged` function calls the `Document.addHandlerAsync` method to register a handler for the `Document.ActiveViewChanged` event.
- To display information, this example uses the `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 to replace the `showNotification` function with your own code.

JavaScript

```
// General Office.onReady function. Called after the add-in loads and Office JS is
// initialized.
Office.onReady(() => {
    // Get whether the current view is edit or read.
    const currentView = getActiveFileView();

    // Register the active view changed handler.
    registerActiveViewChanged();

    // Render the content based off of the current view.
    if (currentView === Office.ActiveView.Read) {
        // Handle read view.
        console.log('Current view is read.');
```

// You can add any specific logic for the read view here.

```
    } else {
        // Handle edit view.
        console.log('Current view is edit.');
```

// You can add any specific logic for the edit view here.

```
    }
});

// Gets the active file view.
function getActiveFileView() {
    console.log('Getting active file view...');
    Office.context.document.getActiveViewAsync(function (result) {
        if (result.status === Office.AsyncResultStatus.Succeeded) {
            console.log('Active view:', result.value);
            return result.value;
        } else {
```

```

        console.error('Error getting active view:', result.error.message);
        showNotification('Error:', result.error.message);
        return null;
    }
});
}

// Registers the ActiveViewChanged event.
function registerActiveViewChanged() {
    console.log('Registering ActiveViewChanged event handler...');
    Office.context.document.addHandlerAsync(
        Office.EventType.ActiveViewChanged,
        activeViewHandler,
        function (result) {
            if (result.status === Office.AsyncResultStatus.Failed) {
                console.error('Failed to register active view changed handler:',
result.error.message);
                showNotification('Error:', result.error.message);
            } else {
                console.log('Active view changed handler registered
successfully.');
```

```

            }
        });
    }

// ActiveViewChanged event handler.
function activeViewHandler(eventArgs) {
    console.log('Active view changed:', JSON.stringify(eventArgs));
    showNotification('Active view changed', `The active view has changed to:
${eventArgs.activeView}`);
    // You can add logic here based on the new active view.
}

```

See also

- [Developing Office Add-ins](#)
- [Learn about the Microsoft 365 Developer Program](#) [↗](#)
- [PowerPoint quick starts](#)
 - [Build your first PowerPoint content add-in](#)
 - [Build your first PowerPoint task pane add-in](#)
- [PowerPoint Code Samples](#)
- [How to save add-in state and settings per document for content and task pane add-ins](#)
- [Read and write data to the active selection in a document or spreadsheet](#)
- [Get the whole document from an add-in for PowerPoint or Word](#)
- [Use document themes in your PowerPoint add-ins](#)

Build your first PowerPoint task pane add-in

Article • 09/17/2024

In this article, you'll walk through the process of building a PowerPoint task pane add-in.

Prerequisites

- Node.js (the latest LTS version). Visit the [Node.js site](#) to download and install the right version for your operating system.
- 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.

command line

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

⚠ Note

Even if you've previously installed the Yeoman generator, we recommend you update your package to the latest version from npm.

- Office connected to a Microsoft 365 subscription (including Office on the web).

⚠ Note

If you don't already have Office, you might qualify for a Microsoft 365 E5 developer subscription through the [Microsoft 365 Developer Program](#); for details, see the [FAQ](#). Alternatively, you can [sign up for a 1-month free trial](#) or [purchase a Microsoft 365 plan](#).

Create the add-in project

Run the following command to create an add-in project using the Yeoman generator. A folder that contains the project will be added to the current directory.

command line

```
yo office
```

Note

When you run the `yo office` command, you may receive prompts about the data collection policies of Yeoman and the Office Add-in CLI tools. Use the information that's provided to respond to the prompts as you see fit.

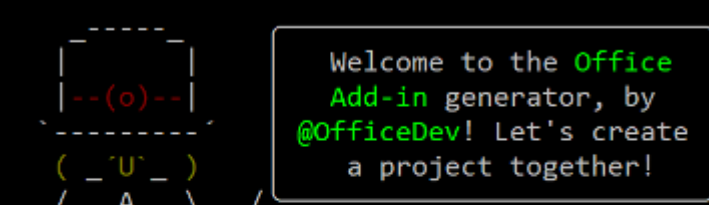
Note

When you run the `yo office` command, you may receive prompts about the data collection policies of Yeoman and the Office Add-in CLI tools. Use the information that's provided to respond to the prompts as you see fit.

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? PowerPoint

```
$ yo office
```



Welcome to the Office Add-in generator, by @OfficeDev! Let's create a project together!

```
? 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? PowerPoint
```

After you complete 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 basic task pane add-in. If you'd like to explore the components of your add-in project, open the project in your code editor and review the files listed below. When you're ready to try out your add-in, proceed to the next section.

- The `./manifest.xml` or `manifest.json` 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 client application.

Try it out

1. Navigate to the root folder of the project.

command line

```
cd "My Office Add-in"
```

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

ⓘ Note

- Office Add-ins should use HTTPS, not HTTP, even while you're developing. If you're prompted to install a certificate after you run one of the following commands, accept the prompt to install the certificate that the Yeoman generator provides. You may also have to run your command prompt or terminal as an administrator for the changes to be made.
- If this is your first time developing an Office Add-in on your machine, you may be prompted in the command line to grant Microsoft Edge WebView a loopback exemption ("Allow localhost loopback for Microsoft Edge WebView?"). When prompted, enter **Y** to allow the exemption. Note that you'll need administrator privileges to allow the exemption. Once allowed, you shouldn't be prompted for an exemption when you sideload Office Add-ins in the future (unless you remove the exemption from your machine). To learn more, see ["We can't open this add-in from localhost" when loading an Office Add-in or using Fiddler.](#)

```
> office-addin-taskpane-js@0.0.1 start
> office-addin-debugging start manifest.xml

Debugging is being started...
App type: desktop
? Allow localhost loopback for Microsoft Edge WebView? (Y/n) Y
```

💡 Tip

If you're testing your add-in on Mac, run the following command before proceeding. When you run this command, the local web server starts.

command line

```
npm run dev-server
```

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

command line

```
npm start
```

- To test your add-in in PowerPoint on a browser, run the following command in the root directory of your project. When you run this command, the local web server starts. Replace "{url}" with the URL of a PowerPoint document on your OneDrive or a SharePoint library to which you have permissions.

⚠ Note

If you are developing on a Mac, enclose the {url} in single quotation marks. Do *not* do this on Windows.

command line

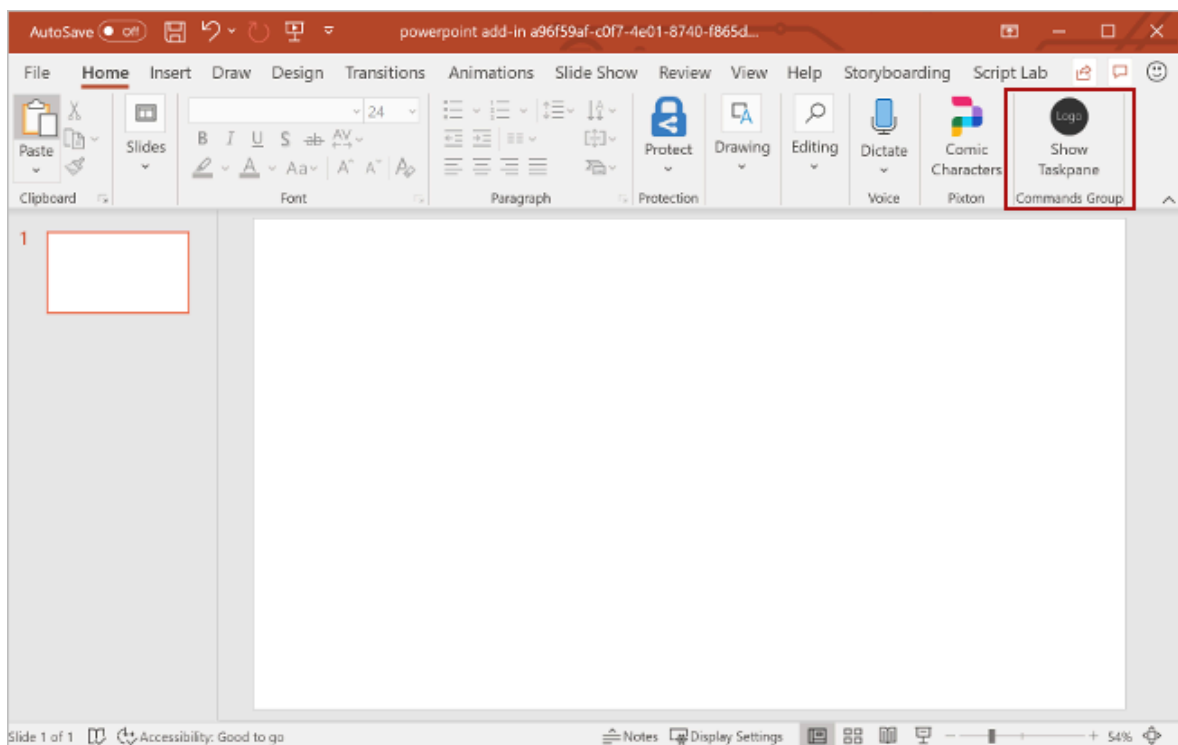
```
npm run start -- web --document {url}
```

The following are examples.

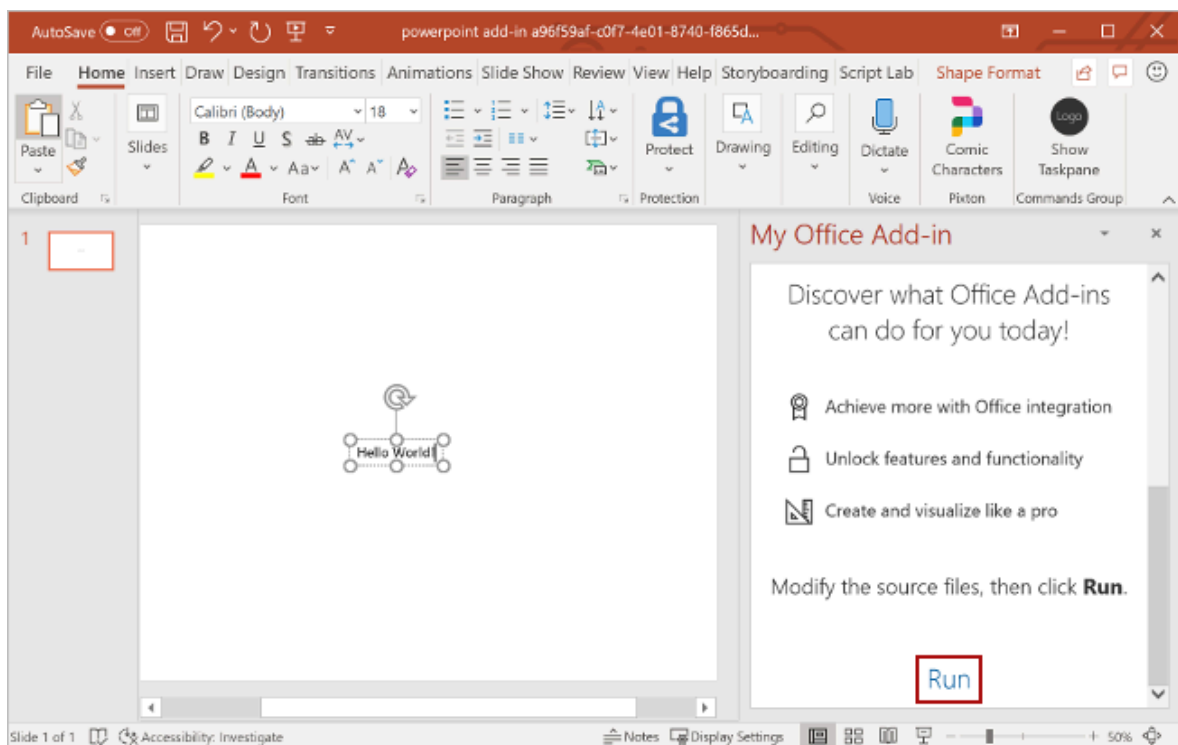
- `npm run start -- web --document https://contoso.sharepoint.com/:t:/g/EZGxP7ksiE5DuxvY638G798BpuhwluxCMfF1WZQj3VYhYQ?e=F4QM1R`
- `npm run start -- web --document https://1drv.ms/x/s!jkcH7spkM4EGgcZUgqthk4IK3N0ypVw?e=Z6G1qp`
- `npm run start -- web --document https://contoso-my.sharepoint-.df.com/:t:/p/user/EQda453DNTpFn11bFPhOVR0BwlrzetbXvnaRYii2lDr_oQ?e=RSccmNP`

If your add-in doesn't sideload in the document, manually sideload it by following the instructions in [Manually sideload add-ins to Office on the web](#).

3. In PowerPoint, insert a new blank slide, choose the **Home** tab, and then choose the **Show Taskpane** button on 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.



5. When you want to stop the local web server and uninstall the add-in, follow the applicable instructions:

- To stop the server, run the following command. If you used `npm start`, the following command also uninstalls the add-in.

command line

```
npm stop
```

- If you manually sideloaded the add-in, see [Remove a sideloaded add-in](#).

Next steps

Congratulations, you've successfully created a PowerPoint task pane add-in! Next, learn more about the capabilities of a PowerPoint add-in and build a more complex add-in by following along with the [PowerPoint add-in tutorial](#).

Troubleshooting

- Ensure your environment is ready for Office development by following the instructions in [Set up your development environment](#).
- Some of the sample code uses ES6 JavaScript. This isn't compatible with [older versions of Office that use the Trident \(Internet Explorer 11\) browser engine](#). For information on how to support those platforms in your add-in, see [Support older](#)

[Microsoft webviews and Office versions](#). If you don't already have a Microsoft 365 subscription to use for development, you might qualify for a Microsoft 365 E5 developer subscription through the [Microsoft 365 Developer Program](#); for details, see the [FAQ](#). Alternatively, you can [sign up for a 1-month free trial](#) or [purchase a Microsoft 365 plan](#).

- The automatic `npm install` step Yo Office performs may fail. If you see errors when trying to run `npm start`, navigate to the newly created project folder in a command prompt and manually run `npm install`. For more information about Yo Office, see [Create Office Add-in projects using the Yeoman Generator](#).

Code samples

- [PowerPoint "Hello world" add-in](#): Learn how to build a simple Office Add-in with only a manifest, HTML web page, and a logo.

See also

- [Office Add-ins platform overview](#)
- [Develop Office Add-ins](#)
- [Using Visual Studio Code to publish](#)

Build your first PowerPoint content add-in

Article • 08/27/2024

In this article, you'll walk through the process of building a PowerPoint [content add-in](#) using Visual Studio.

Prerequisites

- [Visual Studio 2019 or later](#) with the **Office/SharePoint development** workload installed.

ⓘ Note

If you've previously installed Visual Studio, use the Visual Studio Installer to ensure that the **Office/SharePoint development** workload is installed.

- Office connected to a Microsoft 365 subscription (including Office on the web).

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 **Insert content into PowerPoint slides**, 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-formatted add-in only manifest file, which contains all the settings that describe your add-in. These settings help the Office application 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. 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**, find the `<p>` element that contains the text "This example will read the current document selection." and the `<button>` element where the `id` is "get-data-from-selection". Replace these entire elements with the following markup then save the file.

HTML

```
<p class="ms-font-m-plus">This example will get some details about the
current slide.</p>

<button class="Button Button--primary" id="get-data-from-selection">
  <span class="Button-icon"><i class="ms-Icon ms-Icon--plus"></i>
</span>
  <span class="Button-label">Get slide details</span>
  <span class="Button-description">Gets and displays the current
slide's details.</span>
</button>
```

2. Open the file **Home.js** in the root of the web application project. This file specifies the script for the add-in. Find the `getDataFromSelection` function and replace the entire function with the following code then save the file.

JavaScript

```
// Gets some details about the current slide and displays them in a
notification.
```

```
function getDataFromSelection() {
    if (Office.context.document.getSelectedDataAsync) {

Office.context.document.getSelectedDataAsync(Office.CoercionType.SlideRange,

        function (result) {
            if (result.status ===
Office.AsyncResultStatus.Succeeded) {
                showNotification('Some slide details are:', '"' +
JSON.stringify(result.value) + '"');
            } else {
                showNotification('Error:', result.error.message);
            }
        }
    );
} else {
    app.showNotification('Error:', 'Reading selection data is not
supported by this host application.');
```

Update the manifest

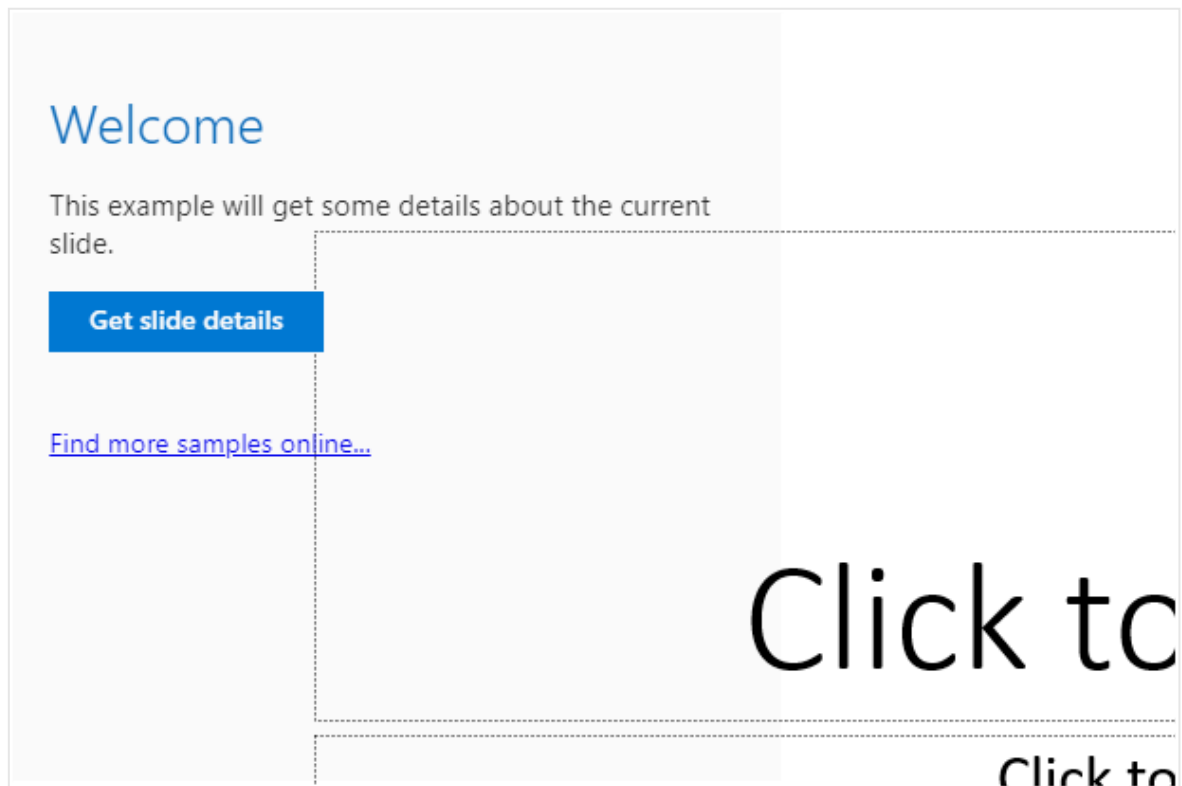
1. Open the add-in only manifest file in the add-in project. This file defines the add-in's settings and capabilities.
2. The `ProviderName` element has a placeholder value. Replace it with your name.
3. The `DefaultValue` attribute of the `DisplayName` element has a placeholder. Replace it with **My Office Add-in**.
4. The `DefaultValue` attribute of the `Description` element has a placeholder. Replace it with **A content add-in for PowerPoint..**
5. Save the file. The updated lines should look like the following code sample.

XML

```
...
<ProviderName>John Doe</ProviderName>
<DefaultLocale>en-US</DefaultLocale>
<!-- The display name of your add-in. Used on the store and various
places of the Office UI such as the add-ins dialog. -->
<DisplayName DefaultValue="My Office Add-in" />
<Description DefaultValue="A content 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 content add-in displayed over the slide.
2. In PowerPoint, choose the **Get slide details** button in the content add-in to get details about the current slide.



ⓘ Note

To see the `console.log` output, you'll need a separate set of developer tools for a JavaScript console. To learn more about F12 tools and the Microsoft Edge DevTools, visit [Debug add-ins using developer tools for Internet Explorer](#), [Debug add-ins using developer tools for Edge Legacy](#), or [Debug add-ins using developer tools in Microsoft Edge \(Chromium-based\)](#).

Next steps

Congratulations, you've successfully created a PowerPoint content add-in! Next, learn more about [developing Office Add-ins with Visual Studio](#).

Troubleshooting

- Ensure your environment is ready for Office development by following the instructions in [Set up your development environment](#).
- Some of the sample code uses ES6 JavaScript. This isn't compatible with [older versions of Office that use the Trident \(Internet Explorer 11\) browser engine](#). For information on how to support those platforms in your add-in, see [Support older Microsoft webviews and Office versions](#). If you don't already have a Microsoft 365 subscription to use for development, you might qualify for a Microsoft 365 E5 developer subscription through the [Microsoft 365 Developer Program](#); for details, see the [FAQ](#). Alternatively, you can [sign up for a 1-month free trial](#) or [purchase a Microsoft 365 plan](#).
- If your add-in shows an error (for example, "This add-in could not be started. Close this dialog to ignore the problem or click "Restart" to try again.") when you press **F5** or choose **Debug > Start Debugging** in Visual Studio, see [Debug Office Add-ins in Visual Studio](#) for other debugging options.

See also

- [Office Add-ins platform overview](#)
- [Develop Office Add-ins](#)
- [Using Visual Studio Code to publish](#)

Tutorial: Create a PowerPoint task pane add-in

Article • 11/21/2024

In this tutorial, you'll create a PowerPoint task pane add-in that:

- ✓ Adds an image to a slide
- ✓ Adds text to a slide
- ✓ Gets slide metadata
- ✓ Adds new slides
- ✓ Navigates between slides

Create the add-in

💡 Tip

If you've already completed the [Build your first PowerPoint task pane add-in](#) quick start using the Yeoman generator, and want to use that project as a starting point for this tutorial, go directly to the [Insert an image](#) section to start this tutorial.

If you want a completed version of this tutorial, visit the [Office Add-ins samples repo on GitHub](#) [↗].

Prerequisites

- Node.js (the latest LTS version). Visit the [Node.js site](#) [↗] to download and install the right version for your operating system.
- 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.

command line

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

ⓘ Note

Even if you've previously installed the Yeoman generator, we recommend you update your package to the latest version from npm.

- Office connected to a Microsoft 365 subscription (including Office on the web).

ⓘ Note

If you don't already have Office, you might qualify for a Microsoft 365 E5 developer subscription through the [Microsoft 365 Developer Program](#) [↗]; for details, see the [FAQ](#). Alternatively, you can [sign up for a 1-month free trial](#) [↗] or [purchase a Microsoft 365 plan](#) [↗].

Create the add-in project

Run the following command to create an add-in project using the Yeoman generator. A folder that contains the project will be added to the current directory.

command line

```
yo office
```

ⓘ Note

When you run the `yo office` command, you may receive prompts about the data collection policies of Yeoman and the Office Add-in CLI tools. Use the information that's provided to respond to the prompts as you see fit.

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?** PowerPoint

```
$ yo office

  Welcome to the Office
  Add-in generator, by
  @OfficeDev! Let's create
  a project together!

? 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? PowerPoint
```

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

Complete setup

1. Navigate to the root directory of the project.

command line

```
cd "My Office Add-in"
```

2. Open your project in VS Code or your preferred code editor.

💡 Tip

On Windows, you can navigate to the root directory of the project via the command line and then enter `code .` to open that folder in VS Code. On Mac, you'll need to [add the code command to the path](#) before you can use that command to open the project folder in VS Code.

Insert an image

Complete the following steps to add code that inserts an image into a slide.

1. Open the project in your code editor.
2. In the root of the project, create a new file named **base64Image.js**.

- Open the file **base64Image.js** and add the following code to specify the Base64-encoded string that represents an image.

JavaScript

```
export const base64Image =  
  
"iVBORw0KGgoAAAANSUgAAZAAAAEFCIAAAABCDiZrAAAACXBIWXMAAAAsSAAALEgHS3X  
78AAAgAE1EQVR42u2dzW9bV3rGn0w5wLBTRpSACAUDmDRowGoj1DdAtBA6suksZmtmV3Qj+  
i8w3XUB00X3pv8CX68Gswq96aKLhI5bCKiM+gpVphIa1qQBcQbyQB/hTJlpOHUXlyEvD885  
vLxfvCSfH7KIjVuUrnif+z7nPOd933v37h0IIWQe+BEvASGEgkUIIRQsQggFixBCKFiEEEL  
BIoRQsAghhIJCCEULEIIBYsQQihYhBBCwSKEULAIYSCRQghFCxCCAWELEIoiWQQQsEihC  
wQCV4CEgDdJvYM9C77f9x8gkyJV4UEznvs6U780rvAfgGdg5EPbr9CyuC1IbSEJGa8KopqB  
WC/gI7Fa0MoWCROHJZw/lxwDl3isITeBa8QoWCRyOk2JR9sVdF+qvwnnQPsF+SaRSEjFCwS  
Cr0LNC04rYkfb5s4vj/h33Y0cFSWy59VlIsGIRQs4pHTGvYMDJvIjup0x5Ir0Tjtp5K/mTK  
wXsSLq2hUWG0R93CXkKg9oL0+ldnFpil+yhlicIM06NA2cXgXySyuV7Fe5CUnFCziyQ02qm  
g8BIDUDWzVkuPiPFHY8xOCGT77EWK84FEZbx4DwOotbJpI5nj5CQWLTOmBj8votuRqBWP8  
KJWABIr2KpLwlmHpeHKff4BsmXxFQmhYBGLBxzoY7Y1ljx0cFAMottS6JH+4Xh69IhEgoW  
cesBNdVQozLyd7whrdrGbSYdIqFgkQkecMD4ep09QB4I46v4tmbtGeK3QYdIKFhE7gEHjO/  
odSzsfRzkS1+5h42q+MG0hf2CuPlIh0goWPSAogcccP2RJHI1riP+kQYdVK9Fh0goWPSAk8  
2a5xCDG4zPJawTxnvSIVKwKFj0gEq1go8QgxtUQQeNZtEhUrB4FZbaA9pIN+98hhcatbNp  
qRoGgRKpdAhUrDIMnpAjVrpJSNApK/uRi7pEC1YZIk84KDGgQ+IBhhicMP6HRg1ycedgVI6  
RELWL14POFCr8VWkszpe3o76G1aFs9ws+dMhUrDIInvAAeMB0ZBCDG6QBh2kgVI6RAoWWRy  
PqBEI9+oQEtKgg3sNpU0kYJGF8oADxgOioUauXKIK0kxV99EhUrDIgnhAG+mCUQqHbpeaNb  
4JgOn3AegQKVhkvj2gjXRLlrIQgxtUQYdPNysOkYJF5tUDarQg4hCDS1u3VZd83IOw0iFSs  
MiceUCNwp3WYH0Wx59R6ls9W1c6RAoWmQ8PaCndz55hiMEN4zsDNhMDpXSIFCwylx5Qo1a9  
C3yVi69a2ajCWZ43N0kQKVgkph5wwHi+KQ4hBs9SC9+RMTpEChaJlwfUFylWEafP5uMKqII  
OPv0sHSIFi8TFAzpLiXxF/KCbdetEGutFUSa6TXQsdKypv42UgZQHfrW0hb06q8nPPqCD/z  
U40kQKFpm9B7SRbrTpQwzJHNaL/VHyiRVF0dfC2xp0zMnKlUgJw0amhGRW/ZM+w5sqzuqTN  
Wtb9nKBZDLoEC1YZGYe0EYaeNWHGDaquHJv5CPnz/H9BTowkjmSfKtD0X0GS22p1ovYNEdU  
r9vCeR3dJlIG1gojn2o8RKPiRX+D0iw6RAoWmYEH1HioiQZqq47VW32dalUlfi1fQf7ByEd  
UQpMpYfOJ46UPcFweKaMSaWyaWL8z/Mibxzzgqe3G4CC6pT4dIwSLReUCNWrkJMDjh8sMSuk  
1d3bReRgB3hy97iS/SEl+5bQ0LqM4B9gvytaptC6kbwz++vD3ZG0r3EBDowUg6RAoWCd0D9  
isXReTKTYghZbhdUB/UYlKV2TSHitZtYc9QrqynDGy/GnGg+4XJr779ShJ0gNdAKR3i/PAj  
XoIze8BGBS+uhqtWAF4VXUWu3G//ORVqdVRiEumhWgFovHT7gB1LnFAvVaJxYZJ+qx/XRuo  
1X0+RFqzPsF/QFZuEgrVcHnDPCGbFylnajN/wAZZvqgpR8Iz0275tTvJnw1/4sORC6C9xWJ  
LoYCKNrbpuR3Jazp/jxdUJmksOivvAfcLsD4LuLfn5h0JhWlVQ+lyNZDFcUl636GY5/Wpy  
zo3FRZ+WBeT1JhpGDVlIMMbJYfYM3Ba4zuXgkUPGBD5B5K16LaJ4/uh/CCDTvDjW4ROxZm4  
gj7+dwZLY24067AkF90tesCaRYdIwaIHDIZMrmSzv2NNTgl4fLlSXw6kjs8pWN+FfHu3n8p  
/xpSBjWrwl0eHSMGiB/TL+h1JnNJ+xtA6MawXh1ogTWA5S5tvLS8vMVUM6s1j+TKZEASjQ6  
RgkVl6wH4pcUM+zs8qBq9WyRyMGoZP+5J0/nzygrrLSkS40NPmNg/vyr1npiQG9+kQKVhkB  
h5woFbSI8EuQwxTkS1j2xoG0zsHeBVcRs1/RNMqyoM0G9WRjAUD4pzD4GhoHjDsMIEqchX4  
8JuUgU1zJN+kSa4D+LnjHfXiqqsa50ejb8J/fs9TAZjFtiXXvgADpaqXZsqUFRY94NRq1ag  
ErFbrRWzVR9Tq9Jl0rWy75NncCf982n+o+sYCDJTSIVKw6AGnRhoQbZsBv3S+MlyxAtC7xP  
F9WMUJDsi5M+gmVCWImpvolorOgXzTMPBAKR0iBWvuPWB4+4CiWj2Rz3MPcFSXHb90Nmawb  
WDLRVZAc2pHZTkF2fWdKugQRqBUCvcQKVj0gI6qRxYQtFvGBIUdvhQ2fmk/VR7fk5Q5jr+2  
fmfygrnPtfM+fu8qa6lEFHcI1lGocolWkQwwcLrr79oBB9YRxg7SDXbDjJISue71LHJWnrno  
+vRh+BX2Xq2Q006+Hf3TTXsYl43M3BhVcZFNjEYvIlUUNvAgrIX1gINqRdpvM0C1EhatbB  
vowaM5ne0Ve/L2VX176/jip88CUysAhyV5SRheoFRSfV+i8RAvckH+XkyweBW8qNWeEe1EP  
1XkKqgQw3j/T3sxyNv6cSKNm02xA3KroVLv1gq4Xh1u3vUusWcE7KESK7jZlHvSoDqU+q/4  
CAUrItoMwtUoRvup1KpRCWxb0KiNqFXvcoreWCem/ETH+ILRYJnvJz1xz+7wrt/19qkuHUI  
IrMk9bxaZeJil1t2mYMWdjoVWFae1sAouVeQq2LUYzWfRaVG1dR9PnKp802EpxG016TCOGZ  
sOb6tk9RayZVZVFkwZ8cff4b/+Htcq8sd17wInJt5UA17SUqnVWR0vbwf5Qn5KgP06bo0mU  
0K2LJetbgtvqjgxQw8uqcbthDH+OrHS/5FV19MuJDXreoSCFQC9C3yxisQK8hVk1dteZ3W8
```


qQY2VFm680F/emj0JNJ430DKQCKN3gU6FrrNSHF9VaMrfI68F+ynXVKpkhxdRyX0TlQzv4
hFKyABWuwMPGR0WxiJ6kdmmbaJu+7gTpPRbgDbZsqJa9/T8AMrvIlNwx/m4Tx+XhY4yC5R
XGGjzRbeHlbd3ZsWQ0+Qp2mth84nFtSBoQtS0M1cobqqCD50BpMovrj/Dpufyk10BXZueKg
yq6KVjEI/bZMF3ef6aErTp2XiOz08UtIe0gCuCoHMWm5MLWyJfK09HTdihdvWPjc+w0J4wv
bJv4KhfF2VIKFnHLM8f4KjfhkF0yh00TN5vYfDJ510wVED0qR7ENv7Sa5SZQmlhB/gF2Xs0
oTdj+06tjz8Dh3Tlbaow9XMNy/153rGGpDIJ+Ycv5bm6bcvVR5YaiPFcy8Kze6s+4lj4VpI
HS1Vv4sORqa09Yr1L5fa5hUbBmLFiDd/am6Soi0LtAqzqyMK9Sq8BDDEQVdMBooDSxgvXih
AV14RfqxgBSsChYcREsmyv3lImtcU5raJs4q8sjV/MYYpgLrj9Sx1P2C/iuiXfL1EYL4GP
ym5/TRQsC1a8BKu/3qFNbL180a9yVKuWUIWzpmKQrnIPBcsrXHQPT+AucXzf701911ahc1T
2FV7tNmEV8fI2t24jI8FLEC52Ysv9wpbAtsVLGNny2+VyFWGFNX+4SWyReYhPKgrWUuAmsU
XiDNNVFKw1sxJBLGyRGVh7L1fFAq5hzeTd38LL27oo0ABpnykSIG766pzWYH3GS0XBWvJr7
yLg8/1F1J1814pk1lXuhM1CaQkJPixN/jvXK1GMpVpa8u7CvSkj9CGshIIV92e7t0vxeBXG
hGFirN6Sp0ZPa5Jw1gfsdEzBWmbGb4BuE4d3JbdKtszHe1j1lZTjsqTBvJtymFCwFpbspRM
77nAouzE+MnnBAiazK++rYZ9Flw4B4mODgrWkpG5I1nHf1gDFrPa1gverNmQc+5jn0L2L/p
DqzoGkN2mArpChFgrWXD3eS5J38KDjJDTKsMG4aaDlrXTjr1UdJkJPtLPcChYBAEmzSqCH0
X8utySZXV65AFBFGezjgULBS1dIwaIf1DzehVVeVZHFiiN/VFEGoZtVtyUxbtwpGDNDb3f
heUH26Z4Nq3bkhW5TKT9dtciqihDtynpWN2mK6RgzS/vemH5QemU9kZF0toHx6Er8VteSTm
WPQ10Za5w4gWRQsFaZD/Yu5APLOhdyvs6XOfqu+faVhF10KsrFwXjRRZHzFOWlumeKbkqr2
xaVUmOdL3IiEPA5ZXmhPn4b2edy1gUrOVh/02uaY/Vu2TEITi1eiCPMrRNnD9XC9Yz0Zgnc
3SFFKx19YPd5oT+Su2nkgQjIw7TklhR7ldMbOBzQldIwVp0xu+Z8SWScY7K8iKLEQf3bFTl
UYZwdZjXVT4zTLrCGD16eAlm6QfdCJZ9WEdYlByjDmG3FU/mRqoJD90EV3+Ga//o5aUPS77
m2QiFrbQm6l24+ok6B+g2R0pj2xWy9SgFa6HV6o74k09Ykx/vNsdlyficfGVkanRIgpV/4E
uw3v/E4xZBMheYYKn2VZ0HcfS0quK6YaaE4/t8U9MSL1N55X4aRedAXouxVZab54Q0ytBtT
nH933KvkIjFwdIEGsaRVjeZEiMOHsurRmWkyTfdlrj1wb1CCtZy+cHT2nSjorotuWbFvMj6
w6/xhxN81xL/G/zsvY7ks384wfdBDHBURRmkB3EmukIBHp0aBVzDm1F55Wa5ffeyZZF4Vs
rILM79eXG6b/5JX7zS8nHt+r92rDz79gvhPPWkCZpF0S9cgTpHf51maFtQSCpTq0o0d1WC
fPQRUyVFGGs7ouKaq5+IJmJdJYv8PLTMfAdj/ojCZDyd5ZMkd7IqKKMsDHqEcGsishYS+oHT
0zvX016v3FQhYBqrV1/EGeCKxw7pkPBomAtGokV8W3dbXq/Z6A4rMNPYe5Wb8mjDPA9SZuu
cOb3Ey9B60VVUH5wwFEZW3Xxg5kSTkxfUmjj/MrCdz7+ovpvc1xYo2HTVKqVz5xtqyo6zfW
il+VIQsGaGz/4xnevBelhHQD5C17eDqA88fCpcX6cns0Fv3JPHmUQWRZ7Y/yYDvcKaQkX2Q
+6P46j5+uS5IN2xCE09C7xrTWbC36toiyOpgq+KS25SVfICmtpyqsTM5ivbA/7HN8Iy1emj
qQKOGu01Ihrj+SfEhD+5mFJ0t85AlQDJrrNwA6Kt01xuZCukIK1sIL1IS+qolGRLJDZEQc/
N6dmxqfmU85dufbTANbpPKCa3wXfa+3Co6JjIWX4coWzWt2jJSRT+EGftc/4nSndlMmWo86
R5ivDg3Xd1ryBVwR8ZCrViDiTACdjrnBaJx7g24CCRCIqrwKv01pVifNKpCPtoZwyRlrQfD
0jm6iJMgQuoEyQUrAWX7B6F8ELVu8S38jMTqYUXS8BZ4ag8VBnGyP7NgQb6z/qMX7ZhV/1e
pGnoyhYMeP/vouRHxzw5rG80V0008CcZrBzEORS0VSooqxQDBz0D6fpuLAWsraI8IPDukYm
E2uF0LfbBTPooQVCIGiIDG0zrEbG7ac8pkPBWiCEwEG3GeLOd/up3IiFXWQ5Xdjx/ZntfKm
iDEC4FR9dIQVrQUhmXQXgSLf5pXem0JE9PDN4/jyAELnnS62JMoTa8P7EpCukYC0EH4QZv5
JiH9YZJ6SISig9MM9i5nZgY1VWQgB3EmXnNh9ZCCRCGaSz4cvYE7VhQjoaSHdUKKODjNYIDzu
KZ19ZZSI76pRJF1oiukYC2CH3TGoBHccRw99mGdcQKPODjN40mz2YTavVRa3G3izeMovoHx
c+wssihYc+8H30Z1Szcq8tBmgKvv8TGDmV3xweC8DtEwPk2HgkXBmm8/eFoLd+1XuH+kCzc
BRhycZtAqzibUDiCxoivyvzuqRjuQqyuf1Ilu/UrDm2Q9G7Jikh3WCKrKcZvDN41BC7X/+Nz
Bq+Nk3yurJZnx6UPT1lap8/oBFFgVrfv1gxILVu5QfnUvmcOWe3y8+CBB0DuRHgvyI1F//C
p9+i7/6Bdbv4E/zuv5/yayyH3QYB3EmVrXCr/jDEu8DCtZ8+sG20YNz+e2n8m27a76ngQ3+
eYDtrlZv9UXqp3+BRMrVP9Fui1/PQiwEwUoZdIUULPrBaZAeoAtqUEXj4SzbOWmiDG0zuuV
C4bcsyDddIQVrDhC043ib1hrMLfRMmSP1+fCP4ITz//4WHUuZ7dpQJ0Vndfr6vHkDXSEFa/
4E68Sc5Tejuns/Mn3dmVY4tU0vg9//J379C/zbTdQ/wN7HcsHSRBlad1mUV3SFFKy5JHVD7
HAS9nEcPefP5YZ0rTDD8BtBBIMKtf/oJwDwP/+N869w/Hf44n3861/iP/4WFy+U/0QTZfB/
EGe9q0yo5bKkFa4MXWE4sKd700VVtxnFcRw9x2X5cs+miRdXXX2Fb62RwRMB5hga/4Df/2o
6+dNEGfwfxLle7ddEnq0wp7WRY9gfljJK27PCih4f0YJDMtmqwrzruIw69C5zVh/8FyG//aT
q10nR18H8QJ1/pq1VmVzKIYCXCPaYrpGDnKx98W4vFN3ZU1ucPr1Xm7JhueE2vEukRKfS8k
do5EDdPPWsfoWBF6gfP6gEvAKcM5Cv9/zI15a0rKZEu5bVeUBGHafi9pbz5/R/E2ai0aHcy
611oTkWkvTi89+7d014Fd49QC3sfyz+183qkwjosBXacba2AfEVcJrd1SHUKR9SmFdxsyjX
uRW6W02vu+eRL5USc/YKvaHvKwPYriZV+kfPy1ZJZ7Iz63D1DuZT5c953rLBi4gcDyYsmc9
g08cmXkk29xAryD3CzqbyNBXVTzbnyE3GIrnrDvF6YpzW/B3Gc247dVl++PRdZ3Za40qf50

rM6N07Boh8U7yKf01a2V028njCeM7GCT750dWupDuv4iThEQ2JFZ119TsRZL478+F+Xhsth
nv2ysPSu6TbzLYc/U7BmgvCm9Bm/ShnYtiRS1TlA4yEaD3H+feQQN5+46imq2q3fqMb62mb
Lyvld/g/iOM8k2mcDBl/Tc5ElFNfJXHQDiilYxIVa3Rm5o3wex0kZ2KqL+3ftp3hxFXsGGH
U0Ktgv4Is0Xt4eytaVe5MrAlXT95Qx9Zj1yNBEGXoXk+c5pwydZR5EGWzXPCjWfBZzVUvxi
cWldwrWbHjXm1xe+Vy92jRH1KpzgLP2P5U3Tz+ojp2TyD5SVyADV9r+wTRYfNFGGVnWC706k
YdTwyZfYqktkS4gytKrDKzxw9EEVWexBSsGaDb3fTRYsP3lRofl65wD7BV1fBGFH302RJbW
rwt0bEzRRBjCHca79UEct3pLI1l0ju60RKXd+cW9F1umzkQV1ukIKVoz8oLME8HkcX6l9vU
vsFyZvJDNv29XC5JdQFVl0fxSf8krFUXlCeZXMlWLn1C3BBY+30BqUb56LrB06QgPWHAUr0
OV2Z49NVUJdoGMNb103iqNq+o7wx0RPV2yqowzd5uSMW7eJPUOymDiQLWc1NL6057/Icr9X
SChY8ypYmnUQvWYNcBPLUK3WEfb4Z0ggUYZuE1YR1meSWmxgBp1r7SrF8VZkdQ5G1h2Tubj
HRyhYS+chO5bfXXan9LhPFTrvBDfHiVWHDRCbiIMmyNBWn24T9rSGr3LKO9HfXygX9Z11nL
ciS7jIb0lHwYpXeew/PcP3DpHSz4xRlVQu+x84N8WcxCHikFjR7QB400dsByBe3pYsLyaz2
H6FTVOuj4PX8lZkveVeIQUrzoI10cQl0hNaxDkrLDfbdon0yMKT+0Mqvcv4Rhw2qsqqx89B
nLM69gx5CZzZxc5ryev6LKEGauJdGCjISlYxK8fnHgcZ72Im01dh1+MtsfL7E70VW1UR/b
LT8wpvn/VY23ZRhxSN3S1jM+DOGuF4b6EcFoAwJV7uNkUk1+Dqt1bkSUU3SyyKFhZU14Zn/
crF826e09iZP9r09S1kcmWR+zb6bOp1/xVh3VmGHHQ7FT6b9k+qJJ6l3hVxJ4h7jY0jpQPt
KlJdWs6D0UWE6QURfiQWB153gpCI7d7Pyyg6B/UDUer39Vb2KpLNCuRxxYV1x+NfHEPjX1V
h3Uwo4jD+h2lmvufiOM85m235ek2cVjCy9uizUysYPMJdn6QLT8rWcI0HbpCCtZ8lFdOd5C
6oSuy7LvIaZGcD/y1AjIlbFsJdY57l97HmqpM1kwiDvryymcDDLUNcrlbpKe1bfFwOfD8e
sns9h80k9s+SmyGMgKGjbwc81ZvT+Rwfh85J3npodcIo2bzb4rPH+O/cIEQRQOFWqe4frj0
xPZfCiVHAY/bDTkHyjLwE6BBjVA05nTld7lH8i+gdbQIx/endp6f3o+LJN7F/hitf//mq6E
hBVWkH7QqVbdpqtK2d4Wj07eFCyfZVD4+GEgz7+1QrQoMBaIbqIw8QoQ1BqBXXyw3adL65
KfpvOFT2fK1l0hRSsOfCD475m05zwdLXvnz0DL66i8VByx3Y0sGcEMDJeoPo7UvVENahCE2
VwcxAnQLpN7Bfw8rZygd/DSHb3CilYMRKS67Xp3sXw/Upu1mopn2KfXzXqGHnNfIPROGwT
WVQM01VveGTuSgiDvoog+cpgT69/4scju8HU9kXj3TWi3M2ryhmcA1rmvexVcSnjntbM5ZC
xaY5YrXsjaS0hY6FRBopA8kcUoauIUnjod8tM0kxpVhC6l0o85ZBoVnKiXgdTeJV09iojvy
+vM2nEC6vPaOEa1gUrNAFq220pNWPyl5GeAqa5Z7z52hUAh5o0kAY/DOgbeLwbmj16h0Yak
/tcyJOYDWggY1qf9vUw6I7xqbpnNZgfUbBoiWM3A96a89wWJrabpw+w8vb2C+EpVZQr75nS
iFGHRRhrYZC7Wy6+j9AqzPvKRzB3WZc7WRrpAVVhRc/AvSPxOfk37sxnoRawUkc0ikJR6w
28J5Hwd1nNYiGgm1/Up+cigka3blnq4/xLzMTPT2wx6WkCmxwqJghcnvj/DTDXElItgVk/c
NAPjWms3Q0jtbr6oKA/5h1eNdAbSqOL6/UG+exMrI6udpDYk0BYuCFsZ//B3+5M/6/9+7wF
e5IPNBmUG1sBJsehPA9Ue6iTgLeW2FvHHHcttEiDjGpZrBmqFIKa1xhPVYZ1gIw6a+v0I4
iBOPBEie1QrCtbM3nwLQ+dAua6cLQfWxeEjU/mpbhONh4t5bdtPOZ6egjULuk1f0lJjjqrp
eyLtfYC7k9VburWbwCNmfM5RsFheLbQcqyfrCJMTvaFpu9qxIj2IEz0nJu8eClb0tf2iv+1
Uh3Xgu1XWlXU6TqpH5QW/sOfPAztQRcEiruhYvqalzgW9S3yjsGZrBe/9BhIruK22fGf1uC
RFWZ5TsFjVzxlvHitrAc9FluawN3y3bGd5TsEiEt4uzRNStf6dzMkb3enRRxna5uLXrf0K/
SCApkAULOK2n1+k8yITaoGnyqOL2fLUp+E+Mr2II4t0QsHyJVhLhUpH7L4r7pkYZViex8BS
FekULApWpGgm60wVcdCom7N59JLQbXHp3TMJXgK3vOvBqKF3gY6FbhPdJr5rLn5p8HVppJe
Tk+tvV10c90NjF/UgzshNtoKUGr+nkTKGbRqJJ3j42f8Ds4luEx2rr2XfX6BjLdRNqJqsA8
AqTgj967sydJt4cXwh3gypG8M2DKsFAGzJQMGAe2wzdV7v/3/vY143wpJZbFty0Zmo0Jr5X
Qihao2U1+QnOSRz/ZbWdmsgTWiDULDmkt5Fv93VfPlKje40KsrjykJr4HFBn23Lds9ujoa0
gkVfGwtfqXF2mvZVQgcogZi0bKebo2CRBFsVmo7G0gahmv6lsy2v60YoWmuL7ewiftPPyle
qJutA1oJd1SFe9fcXz83ZD5vumlPPXiUURBBpm8Pooz1gZmAr7Lt1YXylZiqXUDf1dnVtZA
IfHTZbN6e67IkVZMvI1lm+UbDiR6uKRkKwDs5HfTI39CPz6Cs10/QGa1L6KIOf4ayzdXNTF
baZXWxUKVUUrBhj7bdJyHt289pw+LvKzUrU40Igz7KoN1VjJub8ybxmV3kK9xJpGDnJ2wd
lX3Fi2LuKzV7f0dlvK3pogzjW4rxdH0ef3H5CvcwKVhZSLeJ43KQrd/j4yuTOeUqs121ae7
YjoXT2tyUk1N51Y9MSHUFa845q6NRCTdtNftfGc9rjgiDIMks8hXuA1KwFojTGo7LUcfZZ+
srI3Nz3/3g6aKP2nITkIK1yLRNHJVnHF6fua/06eZsVYrDYaYr93CtQqmiYC00024jRkZMf
KUtsQM3B8RXLAU3ASlYSydb31Tw5vEcFksh+cqZuznPV20jyhHzFKylpNtEozKXzVXc+8p4
ujkPpG7gepWbgBSspSeCbcRoGA+LzkX3GDdmmZuAsXpc8hLMkrUC1uo4q+Pr0nINYpiLQjJ
b1kX2ySzeIP4yNZOE5tPkMzyYsSlYLzZpFpRsIiaTAnbFvIPph75R4L8Lexi5/WEIdWEgk
UAIJFGvokbTS+jlYlPvm9h5zU2TUYWKFhketnaeY3MLi9GRFL1yZfYq10qKFjEK8kcNk1sv
+qHoUgoFzmLzSfYqj0yQMEiQZAysFXHJ190MwaZuCpJv3D9EXbYv5iCRQJnrYBti9uIgUmV
vYzBIcUAAAIqSURBVAmYLFNiULBIaGRK2GlyG9HfNdZftsVNQAowiYrBNiJlayq4CUjBIjM
yNwnkK9i2uI3oVqq4CUjBIjPG3kbcec1tRPulysL4nJuAFCwSJ9mytxEpWyNF6Ao2n2CnqZ

```
yXQShYZGasFbBV5zZiX6rsTUDmFShYJNbY24jXHy3venxmt39omZuAFCwyH2TLy7iNuH6nv
wLIqaJgkXmzRcu0jWhvAho1bgJSsMg8M9hGXL+zoD9gtp9X4CYgBYssjmwZtUXbRrQPLe80
KVUULLKI2NuIxudzv41obwJuW9wEpGCRRWe920/FPKfr8VfucROQgkWWjExp/rYR7c7FG1V
KFQWLLB+DXszz30a0NwF5aJlQsChb/W3EeMpW6gY3AQkFi4xipx9itY1obwJuW5QqIj5keQ
kIEJuRrhxfSlhhkSlka4YjXTm+lFCwyNREP9KV40sJBYv4sGY/bCNeuRfuC63ewvYrbgISC
hYJQrY2qmFtIw46F6cMXm1CwSIBefhIV44vJRQsEi6BjHTl+FJCwSLR4XmkK8eXEgoWmQ3T
jnTl+FJCwSiZzJDSVQPH15JAee/du3e8CsQX3Sa6Y730pB8khIJFCKElJIQQChYhhFCwCCE
ULEIIoWARQggFixBCwSKEEAoWIYRQsAghFCxCCKFgEUIIBYsQQsEihBAKFiGEULAIIRQsQg
ihYBFCCAWLEELBIOQQChYhhILFS0AIoWARQkjA/D87uqZQTj7xTgAAAABJRu5ErkJggg=="
;
```

4. Open the file `./src/taskpane/taskpane.html`. This file contains the HTML markup for the task pane.
5. Locate the `<body>` element. Replace it with the following markup, then save the file.

HTML

```
<body class="ms-font-m ms-welcome ms-Fabric">
  <!-- TODO2: Update the header node. -->
  <header class="ms-welcome__header ms-bgColor-neutrallighter">
    
    <h1 class="ms-font-su">Welcome</h1>
  </header>
  <section id="sideload-msg" class="ms-welcome__main">
    <h2 class="ms-font-xl">Please <a target="_blank"
href="https://learn.microsoft.com/office/dev/add-ins/testing/test-
debug-office-add-ins#sideload-an-office-add-in-for-
testing">sideload</a> your add-in to see app body.</h2>
  </section>
  <main id="app-body" class="ms-welcome__main" style="display:
none;">
    <div class="padding">
      <!-- TODO1: Create the insert-image button. -->
      <!-- TODO3: Create the insert-text button. -->
      <!-- TODO4: Create the get-slide-metadata button. -->
      <!-- TODO5: Create the add-slides and go-to-slide buttons.
-->
    </div>
  </main>
  <section id="display-msg" class="ms-welcome__main">
    <div class="padding">
      <h3>Message</h3>
      <div id="message"></div>
    </div>
  </section>
</body>
```

6. In the **taskpane.html** file, replace **TODO1** with the following markup. This markup defines the **Insert Image** button that will appear within the add-in's task pane.

HTML

```
<button class="ms-Button" id="insert-image">Insert Image</button><br/>
<br/>
```

7. Open the file **./src/taskpane/taskpane.js**. This file contains the Office JavaScript API code that facilitates interaction between the task pane and the Office client application. Replace the entire contents with the following code and save the file.

JavaScript

```
/*
 * Copyright (c) Microsoft Corporation. All rights reserved. Licensed
 under the MIT license.
 * See LICENSE in the project root for license information.
 */

/* global document, Office */

// TODO1: Import Base64-encoded string for image.
Office.onReady((info) => {
  if (info.host === Office.HostType.PowerPoint) {
    document.getElementById("sideload-msg").style.display = "none";
    document.getElementById("app-body").style.display = "flex";
    // TODO2: Assign event handler for insert-image button.
    // TODO4: Assign event handler for insert-text button.
    // TODO6: Assign event handler for get-slide-metadata button.
    // TODO8: Assign event handlers for add-slides and the four
    navigation buttons.
  }
});

// TODO3: Define the insertImage function.

// TODO5: Define the insertText function.

// TODO7: Define the getSlideMetadata function.

// TODO9: Define the addSlides and navigation functions.

async function clearMessage(callback) {
  document.getElementById("message").innerText = "";
  await callback();
}

function setMessage(message) {
  document.getElementById("message").innerText = message;
}
```

```
// Default helper for invoking an action and handling errors.
async function tryCatch(callback) {
  try {
    document.getElementById("message").innerText = "";
    await callback();
  } catch (error) {
    setMessage("Error: " + error.toString());
  }
}
```

8. In the **taskpane.js** file above the `Office.onReady` function call near the top of the file, replace `TOD01` with the following code. This code imports the variable that you defined previously in the file `./base64Image.js`.

JavaScript

```
import { base64Image } from "../base64Image";
```

9. In the **taskpane.js** file, replace `TOD02` with the following code to assign the event handler for the **Insert Image** button.

JavaScript

```
document.getElementById("insert-image").onclick = () =>
clearMessage(insertImage);
```

10. In the **taskpane.js** file, replace `TOD03` with the following code to define the `insertImage` function. This function uses the Office JavaScript API to insert the image into the document. Note:

- The `coercionType` option that's specified as the second parameter of the `setSelectedDataAsync` request indicates the type of data being inserted.
- The `asyncResult` object encapsulates the result of the `setSelectedDataAsync` request, including status and error information if the request failed.

JavaScript

```
function insertImage() {
  // Call Office.js to insert the image into the document.
  Office.context.document.setSelectedDataAsync(
    base64Image,
    {
      coercionType: Office.CoercionType.Image
    },
  ),
}
```

```
(asyncResult) => {  
    if (asyncResult.status === Office.AsyncResultStatus.Failed) {  
        setMessage("Error: " + asyncResult.error.message);  
    }  
}  
);  
}
```

11. Save all your changes to the project.

Test the add-in

1. Navigate to the root folder of the project.

command line

```
cd "My Office Add-in"
```

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

ⓘ Note

- Office Add-ins should use HTTPS, not HTTP, even while you're developing. If you're prompted to install a certificate after you run one of the following commands, accept the prompt to install the certificate that the Yeoman generator provides. You may also have to run your command prompt or terminal as an administrator for the changes to be made.
- If this is your first time developing an Office Add-in on your machine, you may be prompted in the command line to grant Microsoft Edge WebView a loopback exemption ("Allow localhost loopback for Microsoft Edge WebView?"). When prompted, enter **y** to allow the exemption. Note that you'll need administrator privileges to allow the exemption. Once allowed, you shouldn't be prompted for an exemption when you sideload Office Add-ins in the future (unless you remove the exemption from your machine). To learn more, see ["We can't open this add-in from localhost" when loading an Office Add-in or using Fiddler](#).


```
> office-addin-taskpane-js@0.0.1 start
> office-addin-debugging start manifest.xml

Debugging is being started...
App type: desktop
? Allow localhost loopback for Microsoft Edge WebView? (Y/n) Y
```

💡 Tip

If you're testing your add-in on Mac, run the following command before proceeding. When you run this command, the local web server starts.

command line

```
npm run dev-server
```

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

command line

```
npm start
```

- To test your add-in in PowerPoint on a browser, run the following command in the root directory of your project. When you run this command, the local web server starts. Replace "{url}" with the URL of a PowerPoint document on your OneDrive or a SharePoint library to which you have permissions.

⚠ Note

If you are developing on a Mac, enclose the {url} in single quotation marks. Do *not* do this on Windows.

command line

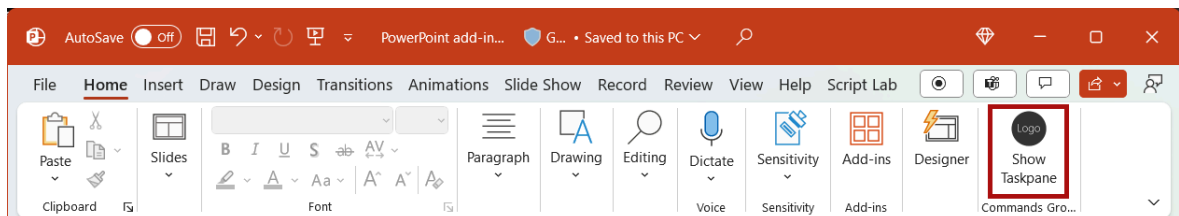
```
npm run start -- web --document {url}
```

The following are examples.

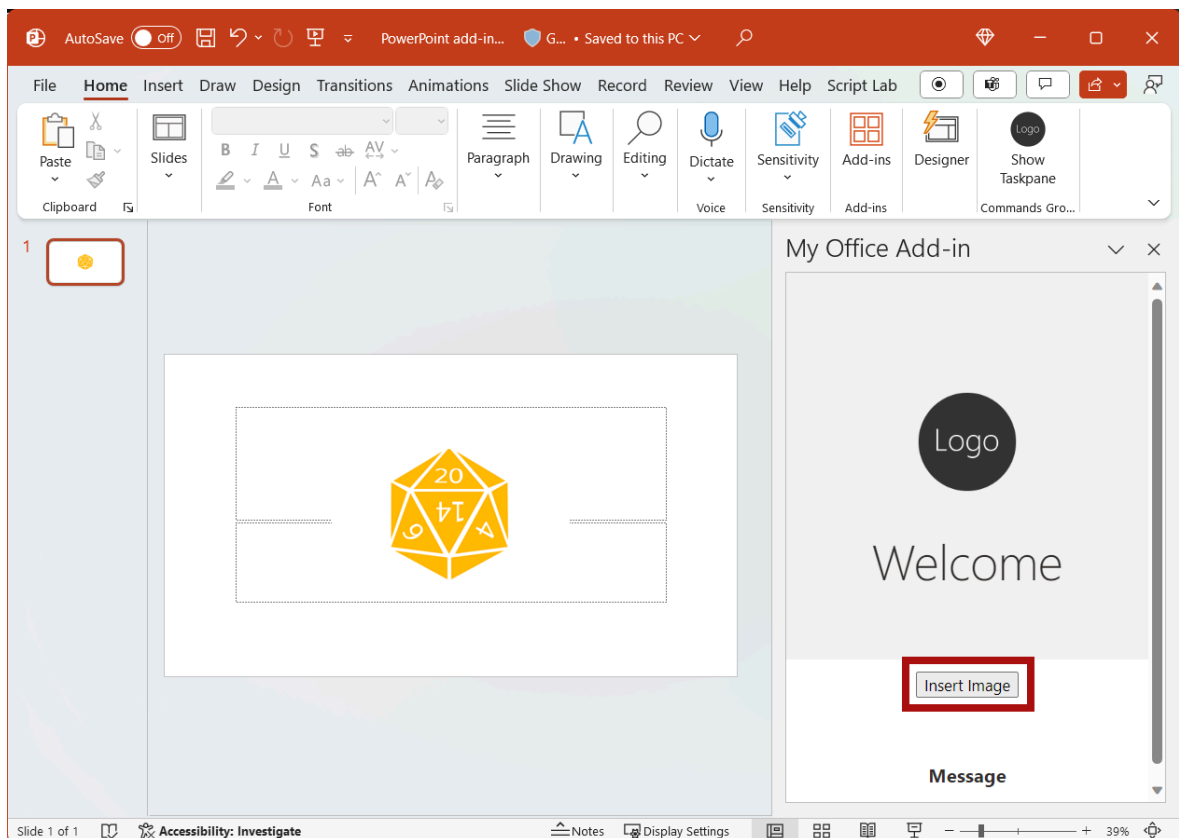
- `npm run start -- web --document https://contoso.sharepoint.com/:t:/g/EZGxP7ksiE5DuxvY638G798BpuhwluxCMfF1WZQj3VYhYQ?e=F4QM1R`
- `npm run start -- web --document https://1drv.ms/x/s!jkcH7spkM4EGgcZUgqthk4IK3N0ypVw?e=Z6G1qp`
- `npm run start -- web --document https://contoso-my.sharepoint-.df.com/:t:/p/user/EQda453DNTpFn11bFPhOVR0BwlrzetbXvnaRYii2lDr_oQ?e=RSccmNP`

If your add-in doesn't sideload in the document, manually sideload it by following the instructions in [Manually sideload add-ins to Office on the web](#).

3. If the add-in task pane isn't already open in PowerPoint, choose the **Home** tab, and then choose the **Show Taskpane** button on the ribbon to open the add-in task pane.



4. In the task pane, choose the **Insert Image** button to add the image to the current slide.



5. When you want to stop the local web server and uninstall the add-in, follow the applicable instructions:

- To stop the server, run the following command. If you used `npm start`, the following command also uninstalls the add-in.

command line

```
npm stop
```

- If you manually sideloaded the add-in, see [Remove a sideloaded add-in](#).

Customize user interface (UI) elements

Complete the following steps to add markup that customizes the task pane UI.

1. In the **taskpane.html** file, replace `TOD02` and the current header section with the following markup to update the header section and title in the task pane. Note:
 - The styles that begin with `ms-` are defined by [Fabric Core in Office Add-ins](#), a JavaScript front-end framework for building user experiences for Office. The **taskpane.html** file includes a reference to the Fabric Core stylesheet.

HTML

```
<header id="content-header">
  <div class="ms-Grid ms-bgColor-neutralPrimary">
    <div class="ms-Grid-row">
      <div class="padding ms-Grid-col ms-u-sm12 ms-u-md12 ms-u-
lg12"> <div class="ms-font-xl ms-fontColor-white ms-fontWeight-
semibold">My PowerPoint add-in</div></div>
    </div>
  </div>
</header>
```

2. Save all your changes to the project.

Test the add-in

1. If the local web server isn't already running, complete the following steps to start the local web server and sideload your add-in.

ⓘ Note

- Office Add-ins should use HTTPS, not HTTP, even while you're developing. If you're prompted to install a certificate after you run one of the following commands, accept the prompt to install the certificate that the Yeoman generator provides. You may also have to run your command prompt or terminal as an administrator for the changes to be made.
- If this is your first time developing an Office Add-in on your machine, you may be prompted in the command line to grant Microsoft Edge WebView a loopback exemption ("Allow localhost loopback for Microsoft Edge WebView?"). When prompted, enter **Y** to allow the exemption. Note that you'll need administrator privileges to allow the exemption. Once allowed, you shouldn't be prompted for an exemption when you sideload Office Add-ins in the future (unless you remove the exemption from your machine). To learn more, see ["We can't open this add-in from localhost" when loading an Office Add-in or using Fiddler.](#)

```
> office-addin-taskpane-js@0.0.1 start
> office-addin-debugging start manifest.xml

Debugging is being started...
App type: desktop
? Allow localhost loopback for Microsoft Edge WebView? (Y/n) Y
```

💡 Tip

If you're testing your add-in on Mac, run the following command before proceeding. When you run this command, the local web server starts.

command line

```
npm run dev-server
```

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

command line

```
npm start
```

- To test your add-in in PowerPoint on a browser, run the following command in the root directory of your project. When you run this command, the local web server starts. Replace "{url}" with the URL of a PowerPoint document on your OneDrive or a SharePoint library to which you have permissions.

! Note

If you are developing on a Mac, enclose the {url} in single quotation marks. Do *not* do this on Windows.

command line

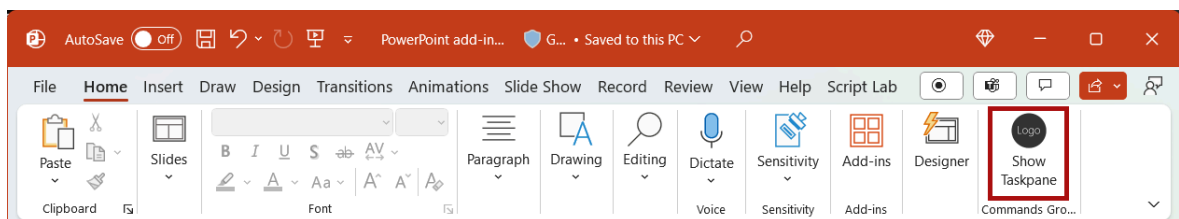
```
npm run start -- web --document {url}
```

The following are examples.

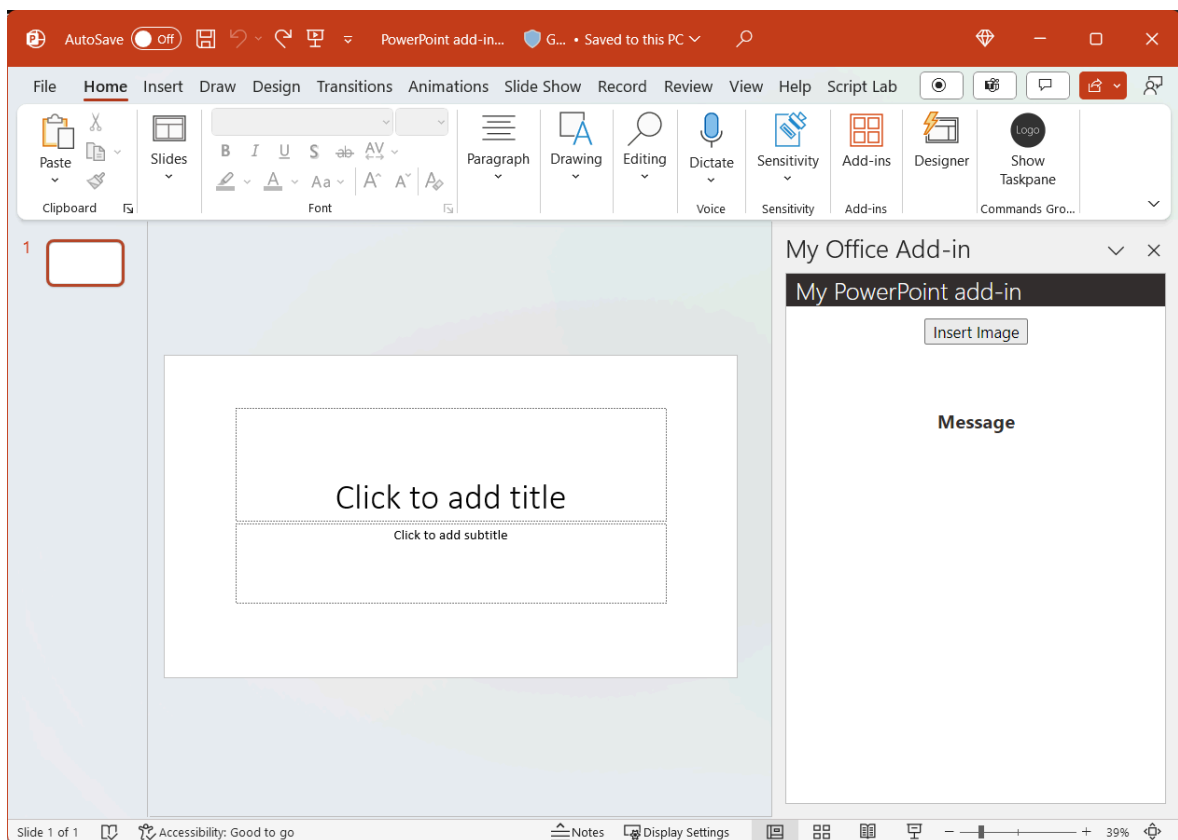
- `npm run start -- web --document https://contoso.sharepoint.com/:t:/g/EZGxP7ksiE5DuxvY638G798BpuhluxCMfF1WZQj3VYhYQ?e=F4QM1R`
- `npm run start -- web --document https://1drv.ms/x/s!jkCH7spkM4EGgcZUgqthk4IK3N0ypVw?e=Z6G1qp`
- `npm run start -- web --document https://contoso-my.sharepoint-df.com/:t:/p/user/EQda453DNTpFn11bFPhOVR0BwlrzetbXvnaRYii2lDr_oQ?e=RSccmNP`

If your add-in doesn't sideload in the document, manually sideload it by following the instructions in [Manually sideload add-ins to Office on the web](#).

2. If the add-in task pane isn't already open in PowerPoint, select the **Show Taskpane** button on the ribbon to open it.



3. Notice that the task pane now contains an updated header section and title.



Insert text

Complete the following steps to add code that inserts text into the title slide which contains an image.

1. In the **taskpane.html** file, replace **TOD03** with the following markup. This markup defines the **Insert Text** button that will appear within the add-in's task pane.

HTML

```
<button class="ms-Button" id="insert-text">Insert Text</button><br/>
<br/>
```

2. In the **taskpane.js** file, replace **TOD04** with the following code to assign the event handler for the **Insert Text** button.

JavaScript

```
document.getElementById("insert-text").onclick = () =>
clearMessage(insertText);
```

3. In the **taskpane.js** file, replace **TOD05** with the following code to define the **insertText** function. This function inserts text into the current slide.

JavaScript

```
function insertText() {  
    Office.context.document.setSelectedDataAsync("Hello World!",  
(asyncResult) => {  
        if (asyncResult.status === Office.AsyncResultStatus.Failed) {  
            setMessage("Error: " + asyncResult.error.message);  
        }  
    });  
}
```

4. Save all your changes to the project.

Test the add-in

1. Navigate to the root folder of the project.

command line

```
cd "My Office Add-in"
```

2. If the local web server isn't already running, complete the following steps to start the local web server and sideload your add-in.

ⓘ Note

- Office Add-ins should use HTTPS, not HTTP, even while you're developing. If you're prompted to install a certificate after you run one of the following commands, accept the prompt to install the certificate that the Yeoman generator provides. You may also have to run your command prompt or terminal as an administrator for the changes to be made.
- If this is your first time developing an Office Add-in on your machine, you may be prompted in the command line to grant Microsoft Edge WebView a loopback exemption ("Allow localhost loopback for Microsoft Edge WebView?"). When prompted, enter **y** to allow the exemption. Note that you'll need administrator privileges to allow the exemption. Once allowed, you shouldn't be prompted for an exemption when you sideload Office Add-ins in the future (unless you remove the

exemption from your machine). To learn more, see ["We can't open this add-in from localhost" when loading an Office Add-in or using Fiddler.](#)

```
> office-addin-taskpane-js@0.0.1 start
> office-addin-debugging start manifest.xml

Debugging is being started...
App type: desktop
? Allow localhost loopback for Microsoft Edge WebView? (Y/n) Y
```

💡 Tip

If you're testing your add-in on Mac, run the following command before proceeding. When you run this command, the local web server starts.

command line

```
npm run dev-server
```

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

command line

```
npm start
```

- To test your add-in in PowerPoint on a browser, run the following command in the root directory of your project. When you run this command, the local web server starts. Replace "{url}" with the URL of a PowerPoint document on your OneDrive or a SharePoint library to which you have permissions.

⚠ Note

If you are developing on a Mac, enclose the {url} in single quotation marks. Do *not* do this on Windows.

command line

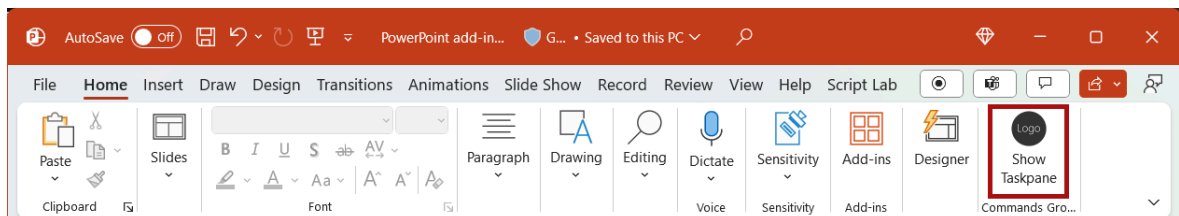
```
npm run start -- web --document {url}
```

The following are examples.

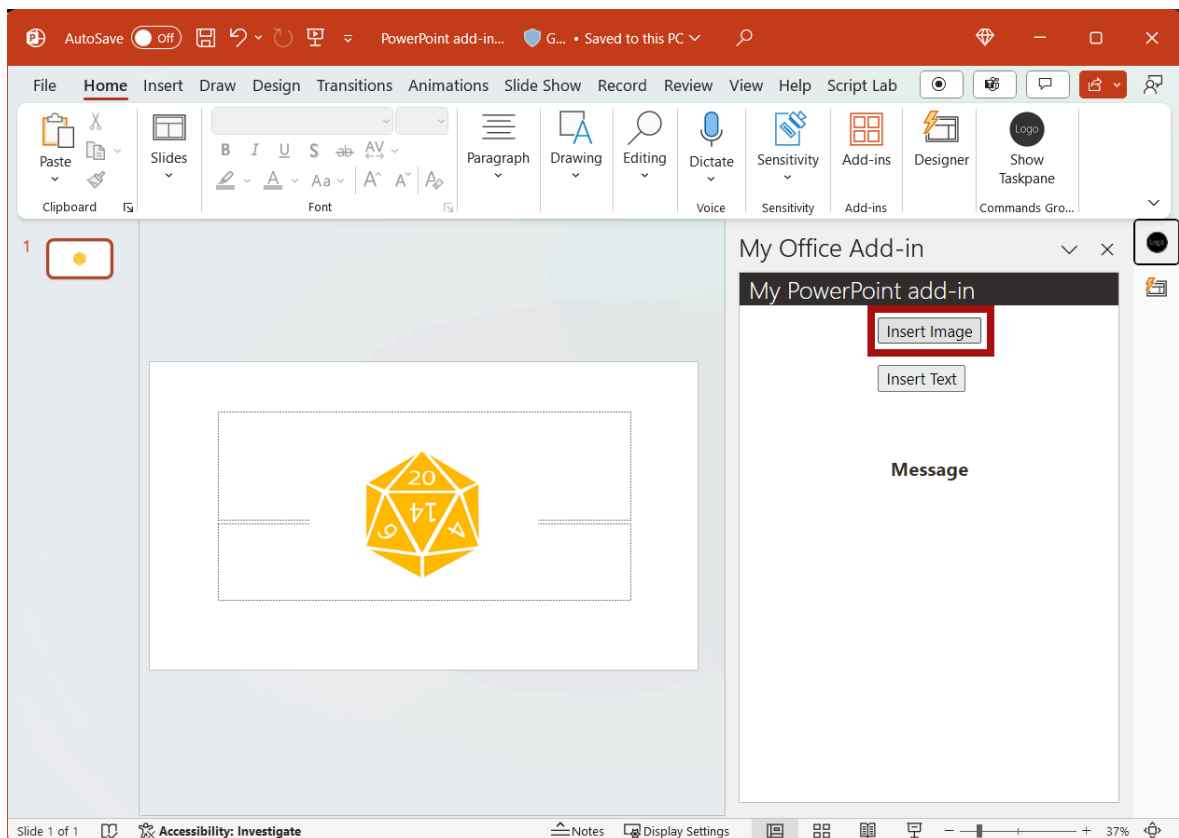
- o `npm run start -- web --document`
`https://contoso.sharepoint.com/:t:/g/EZGxP7ksiE5DuxvY638G798BpuhwluxCMfF1WZQj3VYhYQ?e=F4QM1R`
- o `npm run start -- web --document`
`https://1drv.ms/x/s!jkcH7spkM4EGgcZUgqthk4IK3N0ypVw?e=Z6G1qp`
- o `npm run start -- web --document https://contoso-my.sharepoint-`
`df.com/:t:/p/user/EQda453DNTpFn11bFPhOVR0Bw1rzetbXvnaRYii21Dr_oQ?`
`e=RSccmNP`

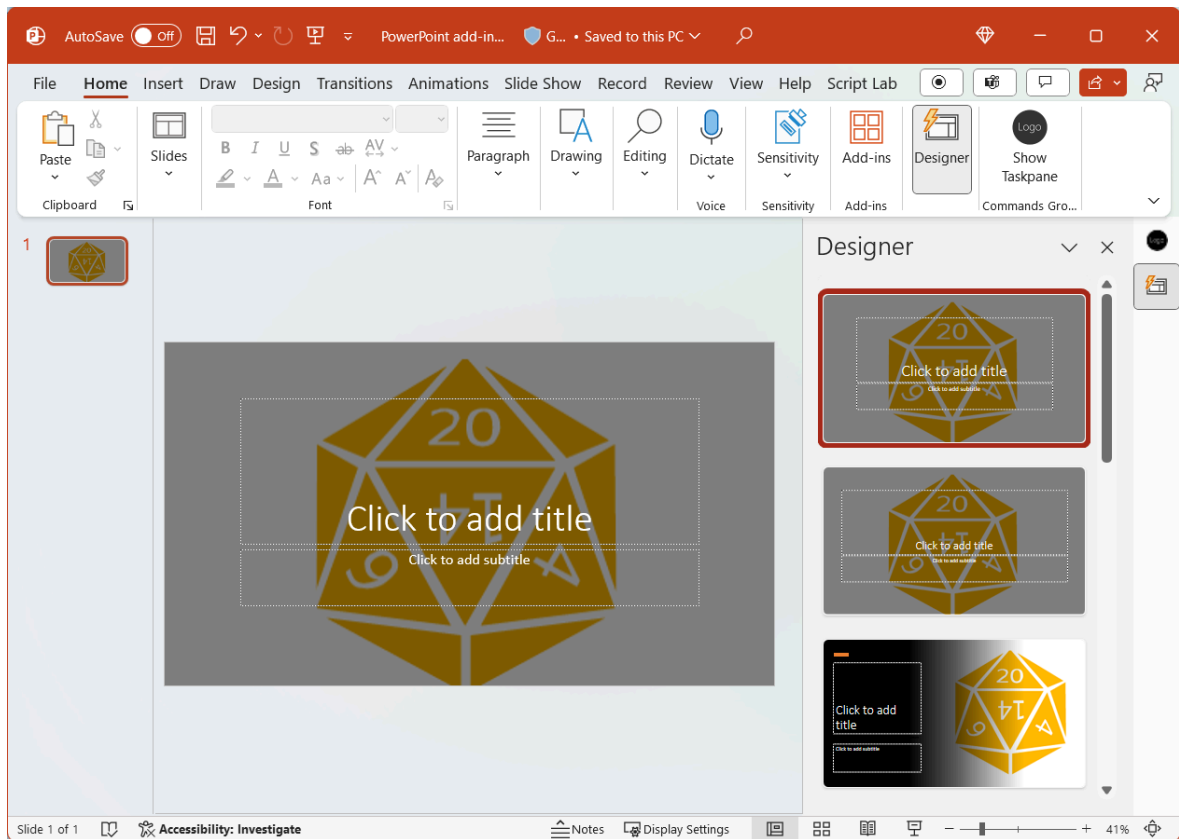
If your add-in doesn't sideload in the document, manually sideload it by following the instructions in [Manually sideload add-ins to Office on the web](#).

3. If the add-in task pane isn't already open in PowerPoint, select the **Show Taskpane** button on the ribbon to open it.

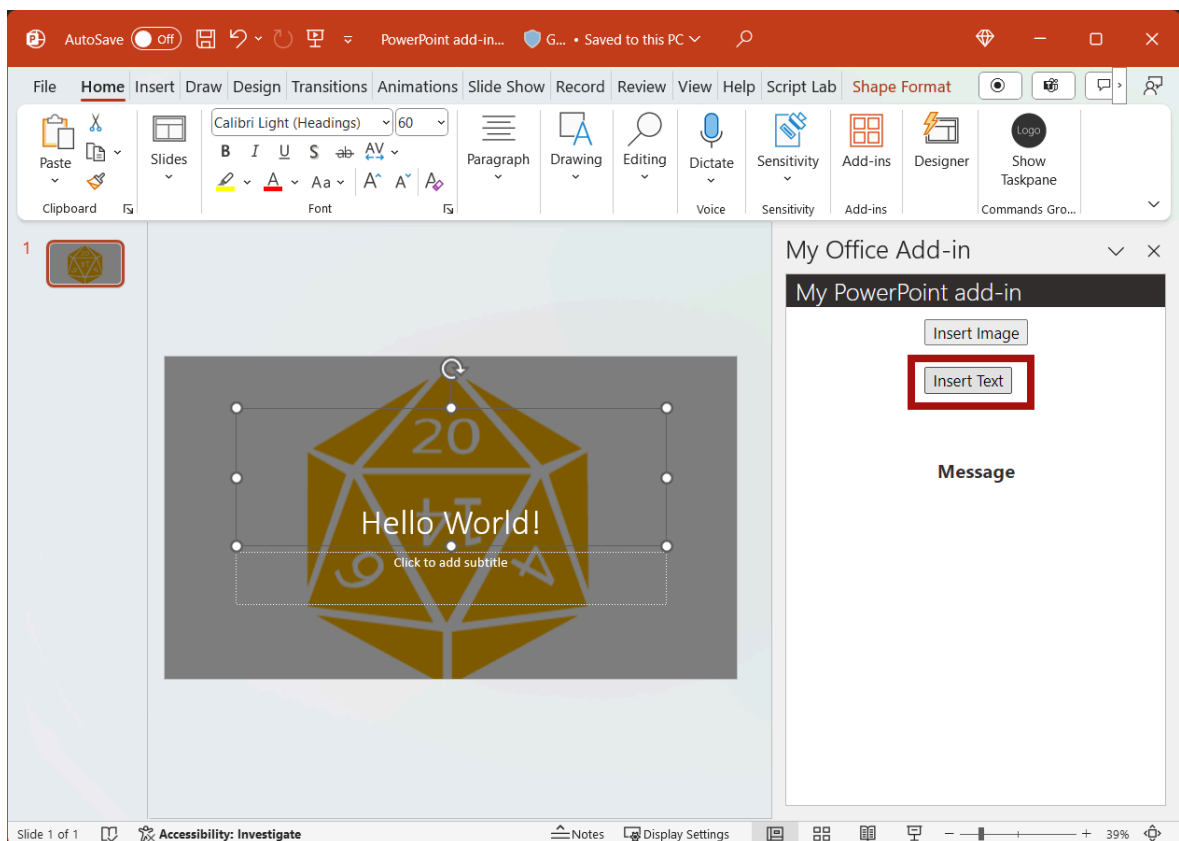


4. In the task pane, choose the **Insert Image** button to add the image to the current slide, then choose a design for the slide that contains a text box for the title.





5. Put your cursor in the text box on the title slide and then in the task pane, choose the **Insert Text** button to add text to the slide.



Get slide metadata

Complete the following steps to add code that retrieves metadata for the selected slide.

1. In the **taskpane.html** file, replace **TODO4** with the following markup. This markup defines the **Get Slide Metadata** button that will appear within the add-in's task pane.

HTML

```
<button class="ms-Button" id="get-slide-metadata">Get Slide  
Metadata</button><br/><br/>
```

2. In the **taskpane.js** file, replace **TODO6** with the following code to assign the event handler for the **Get Slide Metadata** button.

JavaScript

```
document.getElementById("get-slide-metadata").onclick = () =>  
clearMessage(getSlideMetadata);
```

3. In the **taskpane.js** file, replace **TODO7** with the following code to define the **getSlideMetadata** function. This function retrieves metadata for the selected slides and writes it to the Message section in the add-in task pane.

JavaScript

```
function getSlideMetadata() {  
  
Office.context.document.getSelectedDataAsync(Office.CoercionType.SlideRange, (asyncResult) => {  
    if (asyncResult.status === Office.AsyncResultStatus.Failed) {  
        setMessage("Error: " + asyncResult.error.message);  
    } else {  
        setMessage("Metadata for selected slides: " +  
JSON.stringify(asyncResult.value));  
    }  
});  
}
```

4. Save all your changes to the project.

Test the add-in

1. Navigate to the root folder of the project.

command line

```
cd "My Office Add-in"
```

2. If the local web server isn't already running, complete the following steps to start the local web server and sideload your add-in.

⚠ Note

- Office Add-ins should use HTTPS, not HTTP, even while you're developing. If you're prompted to install a certificate after you run one of the following commands, accept the prompt to install the certificate that the Yeoman generator provides. You may also have to run your command prompt or terminal as an administrator for the changes to be made.
- If this is your first time developing an Office Add-in on your machine, you may be prompted in the command line to grant Microsoft Edge WebView a loopback exemption ("Allow localhost loopback for Microsoft Edge WebView?"). When prompted, enter **Y** to allow the exemption. Note that you'll need administrator privileges to allow the exemption. Once allowed, you shouldn't be prompted for an exemption when you sideload Office Add-ins in the future (unless you remove the exemption from your machine). To learn more, see ["We can't open this add-in from localhost" when loading an Office Add-in or using Fiddler.](#)

```
> office-addin-taskpane-js@0.0.1 start
> office-addin-debugging start manifest.xml

Debugging is being started...
App type: desktop
? Allow localhost loopback for Microsoft Edge WebView? (Y/n) Y
```

💡 Tip

If you're testing your add-in on Mac, run the following command before proceeding. When you run this command, the local web server starts.

command line

```
npm run dev-server
```

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

command line

```
npm start
```

- To test your add-in in PowerPoint on a browser, run the following command in the root directory of your project. When you run this command, the local web server starts. Replace "{url}" with the URL of a PowerPoint document on your OneDrive or a SharePoint library to which you have permissions.

ⓘ Note

If you are developing on a Mac, enclose the {url} in single quotation marks. Do *not* do this on Windows.

command line

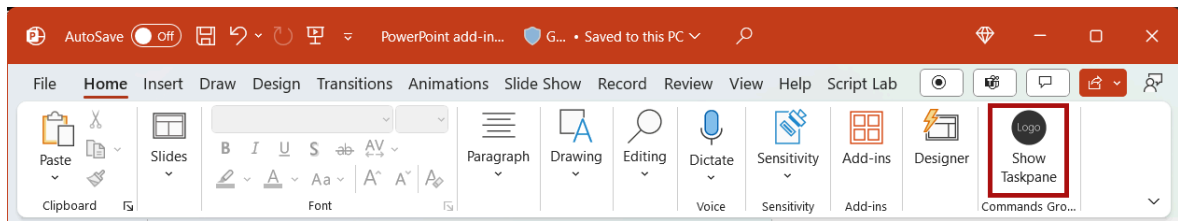
```
npm run start -- web --document {url}
```

The following are examples.

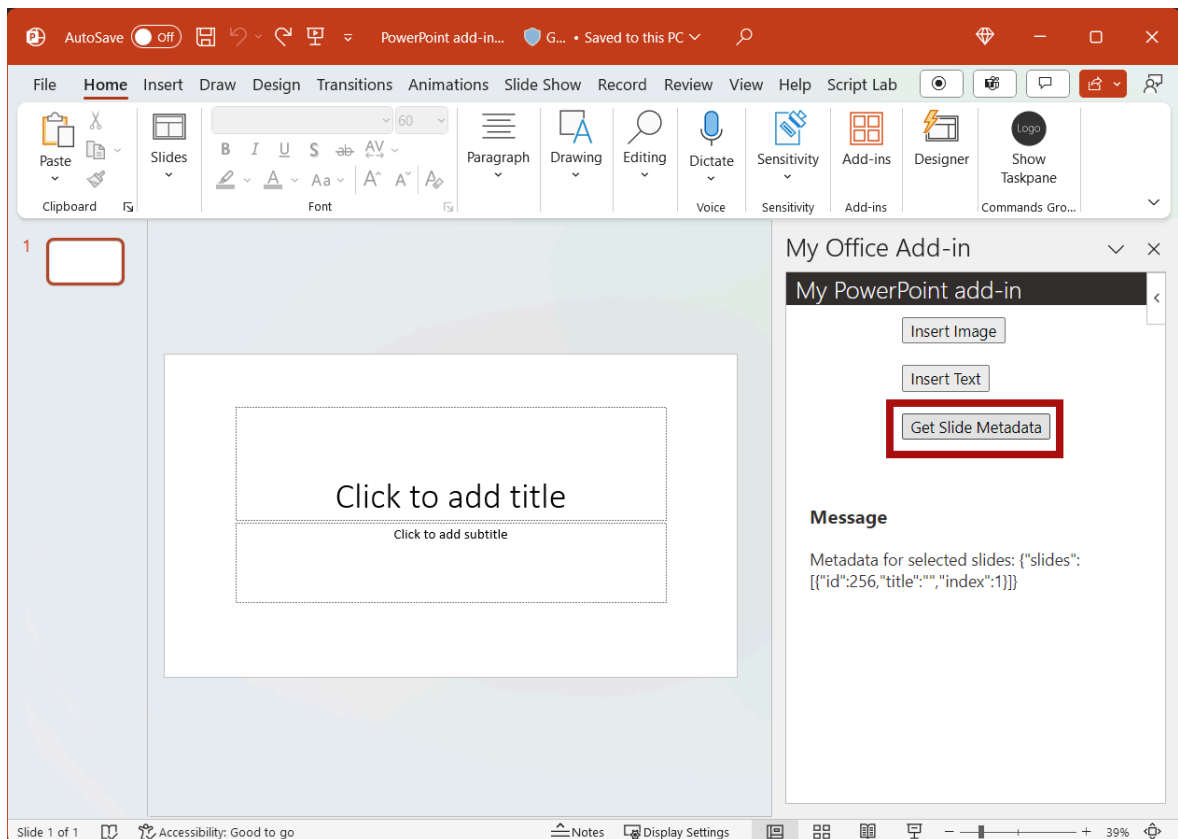
- `npm run start -- web --document https://contoso.sharepoint.com/:t:/g/EZGxP7ksiE5DuxvY638G798BpuhwluxCMfF1WZQj3VYhYQ?e=F4QM1R`
- `npm run start -- web --document https://1drv.ms/x/s!jkcH7spkM4EGgcZUgqthk4IK3N0ypVw?e=Z6G1qp`
- `npm run start -- web --document https://contoso-my.sharepoint-df.com/:t:/p/user/EQda453DNTpFn11bFPhOVR0BwlrzetbXvnaRYii2lDr_oQ?e=RSccmNP`

If your add-in doesn't sideload in the document, manually sideload it by following the instructions in [Manually sideload add-ins to Office on the web](#).

3. If the add-in task pane isn't already open in PowerPoint, select the **Show Taskpane** button on the ribbon to open it.



4. In the task pane, choose the **Get Slide Metadata** button to get the metadata for the selected slide. The slide metadata is written in the Message section below the buttons in the task pane. In this case, the `slides` array within the JSON metadata contains one object that specifies the `id`, `title`, and `index` of the selected slide. If multiple slides had been selected when you retrieved slide metadata, the `slides` array within the JSON metadata would contain one object for each selected slide.



Navigate between slides

Complete the following steps to add code that navigates between the slides of a document.

1. In the `taskpane.html` file, replace `TOD05` with the following markup. This markup defines the four navigation buttons that will appear within the add-in's task pane.

HTML

```
<button class="ms-Button" id="add-slides">Add Slides</button><br><br>
<button class="ms-Button" id="go-to-first-slide">Go to First
```

```

Slide</button><br/><br/>
<button class="ms-Button" id="go-to-next-slide">Go to Next
Slide</button><br/><br/>
<button class="ms-Button" id="go-to-previous-slide">Go to Previous
Slide</button><br/><br/>
<button class="ms-Button" id="go-to-last-slide">Go to Last
Slide</button><br/><br/>

```

2. In the **taskpane.js** file, replace **TODO8** with the following code to assign the event handlers for the **Add Slides** and four navigation buttons.

JavaScript

```

document.getElementById("add-slides").onclick = () =>
tryCatch(addSlides);
document.getElementById("go-to-first-slide").onclick = () =>
clearMessage(goToFirstSlide);
document.getElementById("go-to-next-slide").onclick = () =>
clearMessage(goToNextSlide);
document.getElementById("go-to-previous-slide").onclick = () =>
clearMessage(goToPreviousSlide);
document.getElementById("go-to-last-slide").onclick = () =>
clearMessage(goToLastSlide);

```

3. In the **taskpane.js** file, replace **TODO9** with the following code to define the **addSlides** and navigation functions. Each of these functions uses the **goToByIdAsync** method to select a slide based upon its position in the document (first, last, previous, and next).

JavaScript

```

async function addSlides() {
  await PowerPoint.run(async function (context) {
    context.presentation.slides.add();
    context.presentation.slides.add();

    await context.sync();

    goToLastSlide();
    setMessage("Success: Slides added.");
  });
}

function goToFirstSlide() {
  Office.context.document.goToByIdAsync(Office.Index.First,
Office.GoToType.Index, (asyncResult) => {
    if (asyncResult.status === Office.AsyncResultStatus.Failed) {
      setMessage("Error: " + asyncResult.error.message);
    }
  });
}

```

```

}

function goToLastSlide() {
    Office.context.document.goToByIdAsync(Office.Index.Last,
    Office.GoToType.Index, (asyncResult) => {
        if (asyncResult.status === Office.AsyncResultStatus.Failed) {
            setMessage("Error: " + asyncResult.error.message);
        }
    });
}

function goToPreviousSlide() {
    Office.context.document.goToByIdAsync(Office.Index.Previous,
    Office.GoToType.Index, (asyncResult) => {
        if (asyncResult.status === Office.AsyncResultStatus.Failed) {
            setMessage("Error: " + asyncResult.error.message);
        }
    });
}

function goToNextSlide() {
    Office.context.document.goToByIdAsync(Office.Index.Next,
    Office.GoToType.Index, (asyncResult) => {
        if (asyncResult.status === Office.AsyncResultStatus.Failed) {
            setMessage("Error: " + asyncResult.error.message);
        }
    });
}

```

4. Save all your changes to the project.

Test the add-in

1. Navigate to the root folder of the project.

command line

```
cd "My Office Add-in"
```

2. If the local web server isn't already running, complete the following steps to start the local web server and sideload your add-in.

ⓘ Note

- Office Add-ins should use HTTPS, not HTTP, even while you're developing. If you're prompted to install a certificate after you run one of the following commands, accept the prompt to install the certificate

that the Yeoman generator provides. You may also have to run your command prompt or terminal as an administrator for the changes to be made.

- If this is your first time developing an Office Add-in on your machine, you may be prompted in the command line to grant Microsoft Edge WebView a loopback exemption ("Allow localhost loopback for Microsoft Edge WebView?"). When prompted, enter **Y** to allow the exemption. Note that you'll need administrator privileges to allow the exemption. Once allowed, you shouldn't be prompted for an exemption when you sideload Office Add-ins in the future (unless you remove the exemption from your machine). To learn more, see ["We can't open this add-in from localhost" when loading an Office Add-in or using Fiddler.](#)

```
> office-addin-taskpane-js@0.0.1 start
> office-addin-debugging start manifest.xml

Debugging is being started...
App type: desktop
? Allow localhost loopback for Microsoft Edge WebView? (Y/n) Y
```

💡 Tip

If you're testing your add-in on Mac, run the following command before proceeding. When you run this command, the local web server starts.

command line

```
npm run dev-server
```

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

command line

```
npm start
```

- To test your add-in in PowerPoint on a browser, run the following command in the root directory of your project. When you run this command, the local

web server starts. Replace "{url}" with the URL of a PowerPoint document on your OneDrive or a SharePoint library to which you have permissions.

ⓘ Note

If you are developing on a Mac, enclose the {url} in single quotation marks. Do *not* do this on Windows.

command line

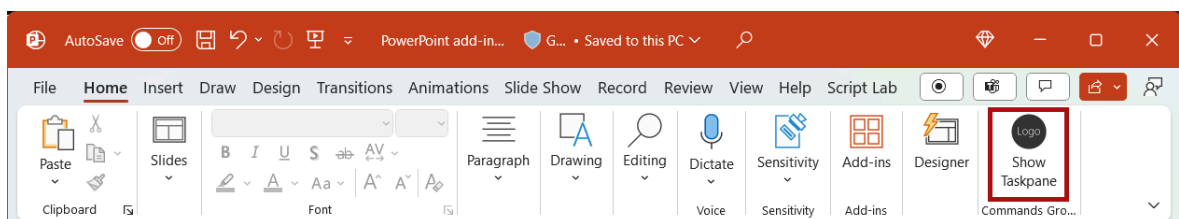
```
npm run start -- web --document {url}
```

The following are examples.

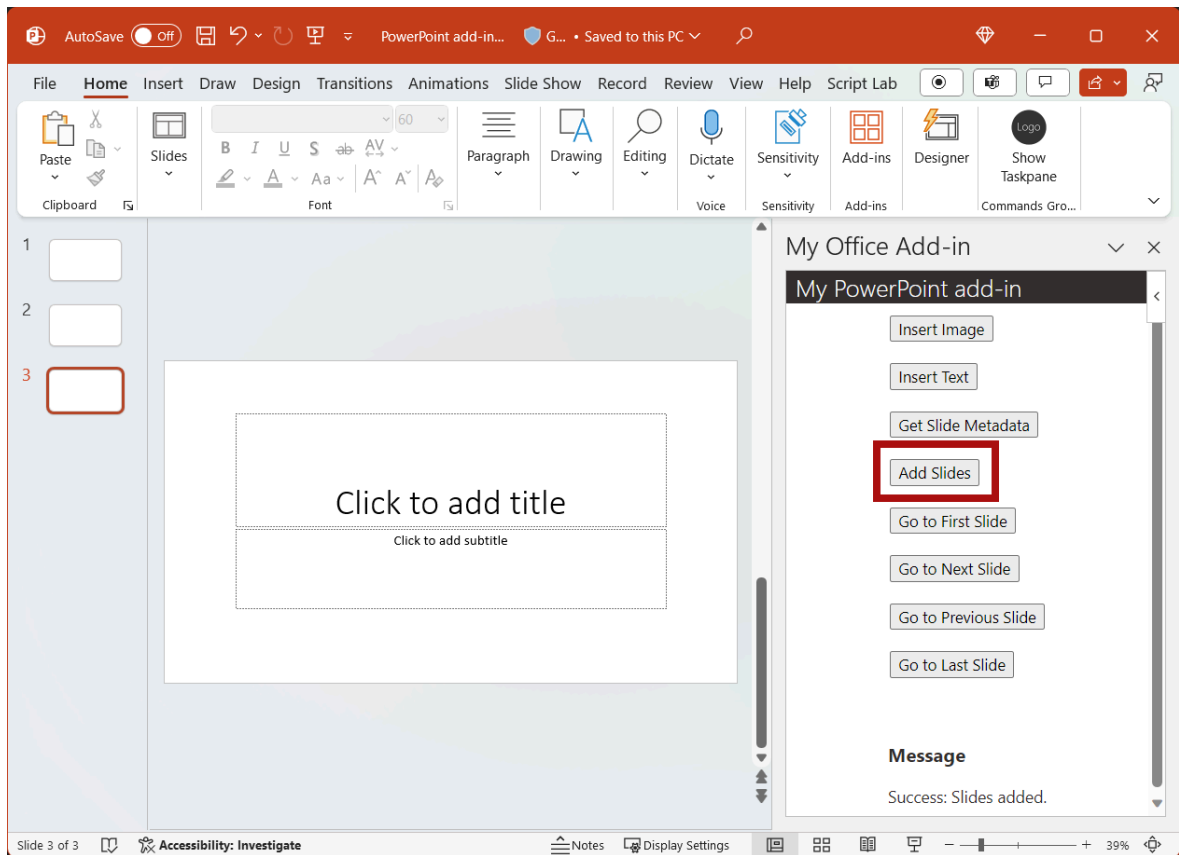
- `npm run start -- web --document https://contoso.sharepoint.com/:t:/g/EZGxP7ksiE5DuxvY638G798BpuhwluxCMfF1wZQj3VYhYQ?e=F4QM1R`
- `npm run start -- web --document https://1drv.ms/x/s!jkcH7spkM4EGgcZUgqthk4IK3N0ypVw?e=Z6G1qp`
- `npm run start -- web --document https://contoso-my.sharepoint-.df.com/:t:/p/user/EQda453DNTpFn11bFPhOVR0BwlrzetbXvnaRYii2lDr_oQ?e=RSccmNP`

If your add-in doesn't sideload in the document, manually sideload it by following the instructions in [Manually sideload add-ins to Office on the web](#).

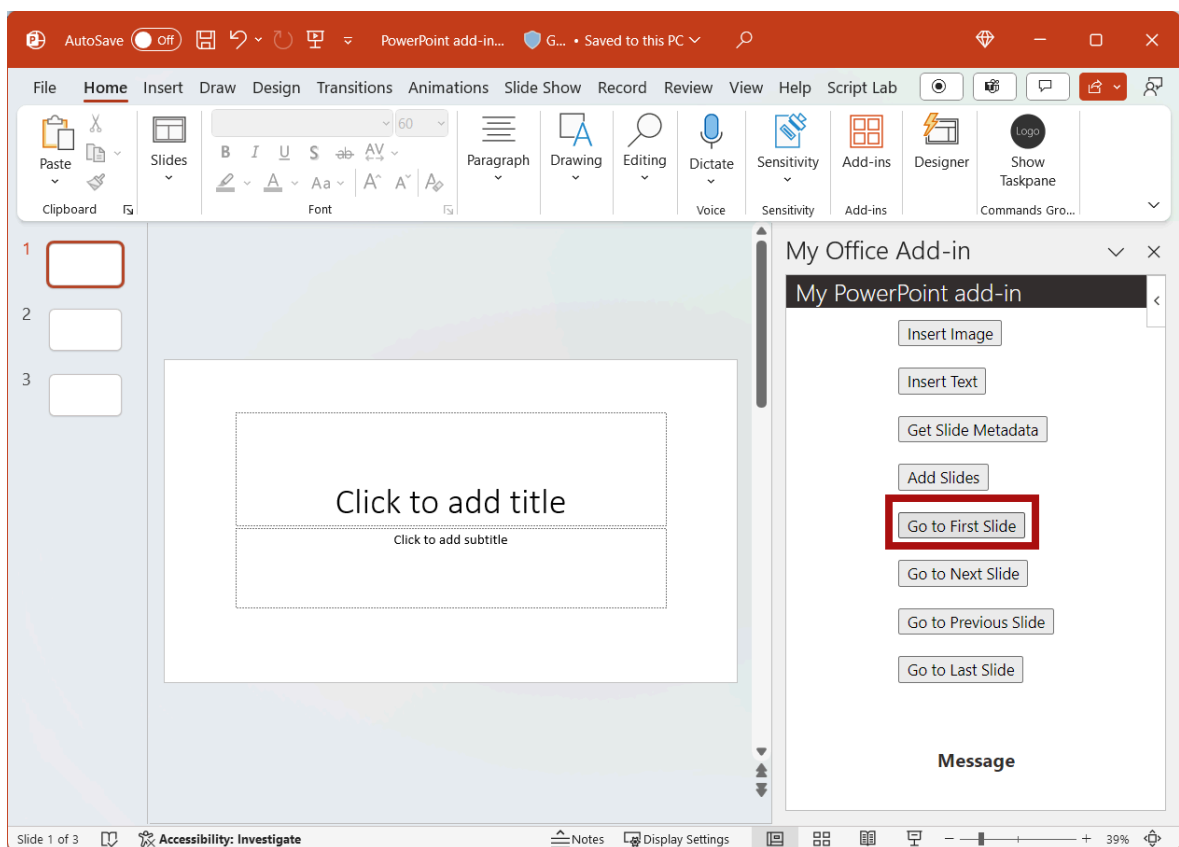
3. If the add-in task pane isn't already open in PowerPoint, select the **Show Taskpane** button on the ribbon to open it.



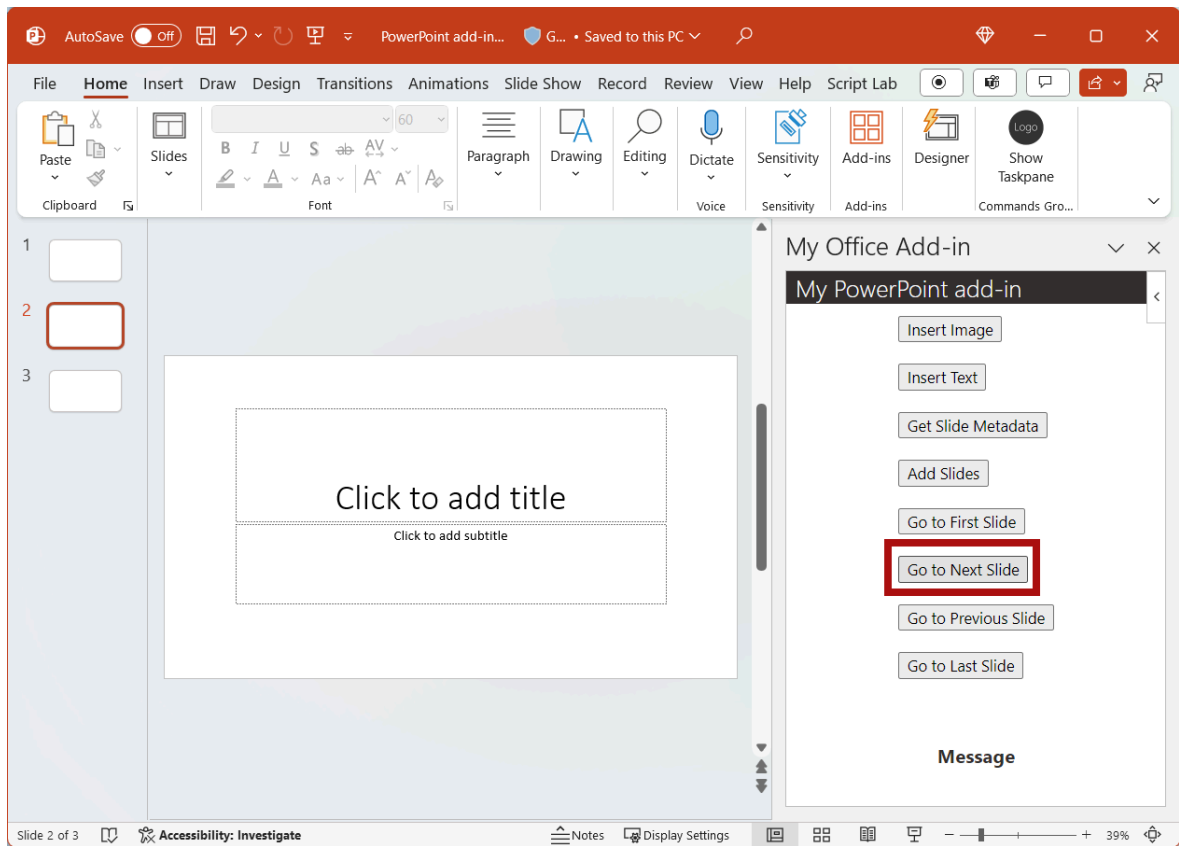
4. In the task pane, choose the **Add Slides** button. Two new slides are added to the document and the last slide in the document is selected and displayed.



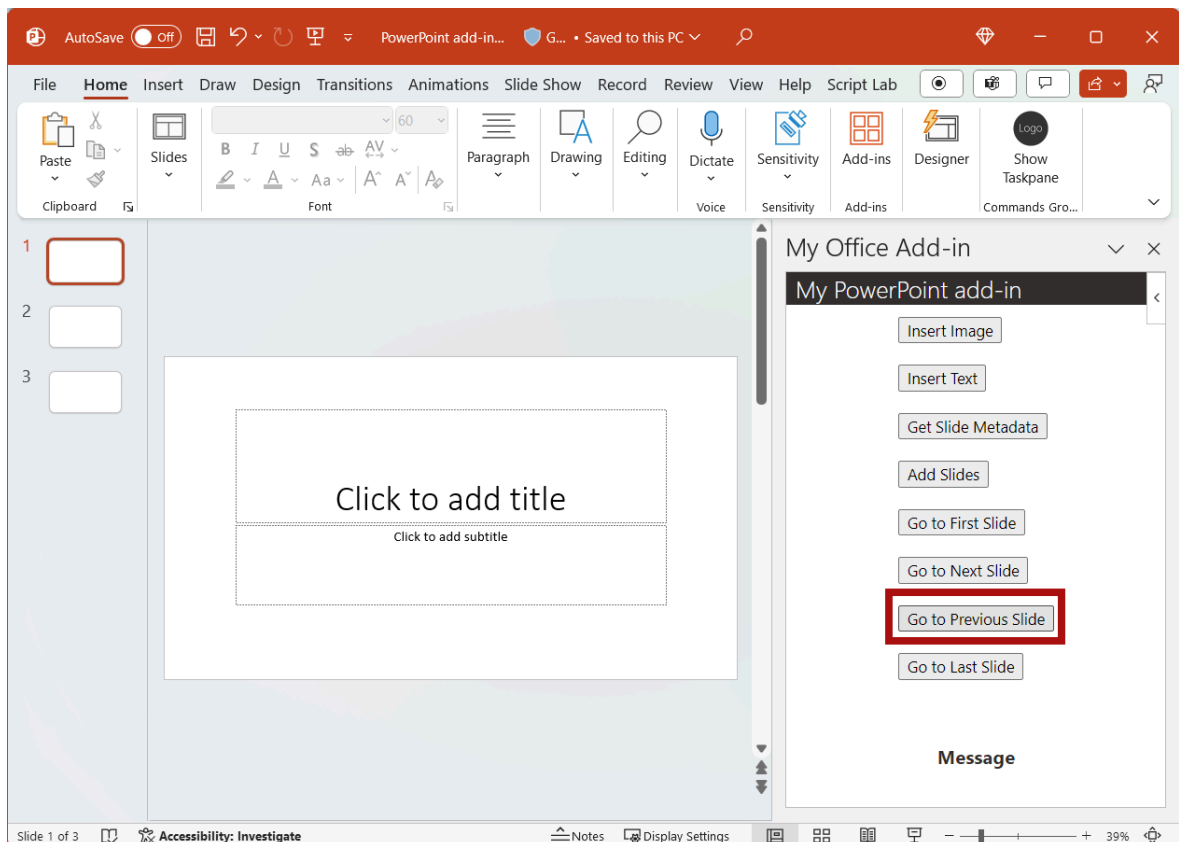
5. In the task pane, choose the **Go to First Slide** button. The first slide in the document is selected and displayed.



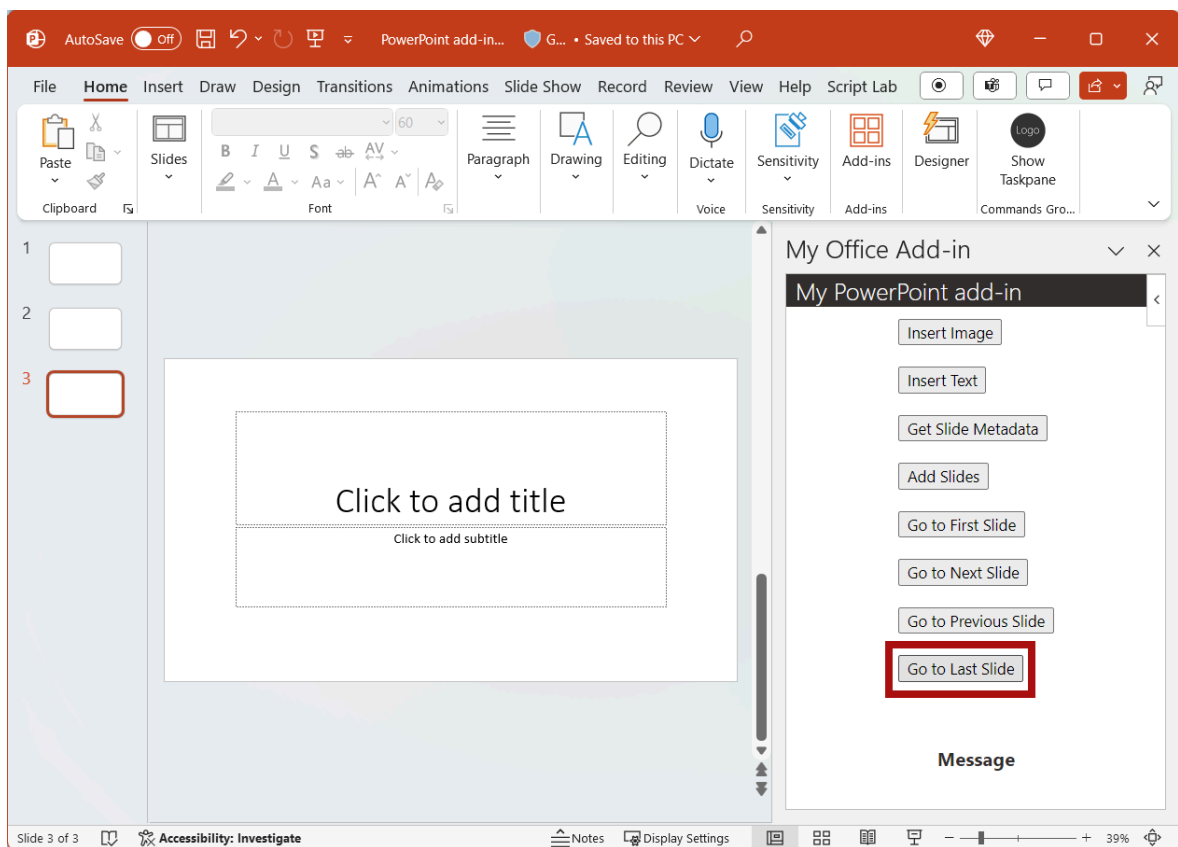
6. In the task pane, choose the **Go to Next Slide** button. The next slide in the document is selected and displayed.



7. In the task pane, choose the **Go to Previous Slide** button. The previous slide in the document is selected and displayed.



8. In the task pane, choose the **Go to Last Slide** button. The last slide in the document is selected and displayed.



9. If the web server is running, run the following command when you want to stop the server.

command line

```
npm stop
```

Code samples

- [Completed PowerPoint add-in tutorial](#): The result of completing this tutorial.

Next steps

In this tutorial, you created a PowerPoint add-in that inserts an image, inserts text, gets slide metadata, and navigates between slides. To learn more about building PowerPoint add-ins, continue to the following article.

[PowerPoint add-ins overview](#)

See also

- [Office Add-ins platform overview](#)

- Develop Office Add-ins

JavaScript API for PowerPoint

Article • 05/30/2025

A PowerPoint add-in interacts with objects in PowerPoint by using the Office JavaScript API, which includes two JavaScript object models:

- **PowerPoint JavaScript API:** The [PowerPoint JavaScript API](#) provides strongly-typed objects that you can use to access objects in PowerPoint. To learn about the asynchronous nature of the PowerPoint JavaScript APIs and how they work with the presentation, see [Using the application-specific API model](#).
- **Common APIs:** The [Common API](#) can be used to access features such as UI, dialogs, and client settings that are common across multiple Office applications. To learn more about using the Common API, see [Common JavaScript API object model](#).

Learn programming concepts

See [PowerPoint add-ins overview](#) for information about important programming concepts.

Learn about API capabilities

For detailed information about the PowerPoint JavaScript API object model, see the [PowerPoint JavaScript API reference documentation](#).

For hands-on experience interacting with content in PowerPoint, complete the [PowerPoint add-in tutorial](#).

Try out code samples in Script Lab

Use [Script Lab](#) to get started quickly with a collection of built-in samples that show how to complete tasks with the API. You can run the samples in Script Lab to instantly see the result in the task pane or document, examine the samples to learn how the API works, and even use samples to prototype your own add-in.

See also

- [PowerPoint add-ins documentation](#)
- [PowerPoint add-ins overview](#)
- [PowerPoint JavaScript API reference](#)
- [Office client application and platform availability for Office Add-ins](#)

- [API Reference documentation](#)

powerpoint package

Classes

 Expand table

PowerPoint.Application	
PowerPoint.Binding	Represents an Office.js binding that is defined in the presentation.
PowerPoint.Binding Collection	Represents the collection of all the binding objects that are part of the presentation.
PowerPoint.Border	Represents the properties for a table cell border.
PowerPoint.Borders	Represents the borders for a table cell.
PowerPoint.Bullet Format	Represents the bullet formatting properties of a text that is attached to the PowerPoint.ParagraphFormat .
PowerPoint.Custom Property	Represents a custom property.
PowerPoint.Custom PropertyCollection	A collection of custom properties.
PowerPoint.Custom XmlPart	Represents a custom XML part object.
PowerPoint.Custom XmlPartCollection	A collection of custom XML parts.
PowerPoint.Custom XmlPartScoped Collection	A scoped collection of custom XML parts. A scoped collection is the result of some operation (such as filtering by namespace). A scoped collection cannot be scoped any further.
PowerPoint.Document Properties	Represents presentation properties.
PowerPoint.Hyperlink	Represents a single hyperlink.
PowerPoint.HyperlinkCollection	Represents a collection of hyperlinks.
PowerPoint.Margins	Represents the margins of a table cell.
PowerPoint.Page Setup	Represents the page setup information for the presentation.

PowerPoint.ParagraphFormat	Represents the paragraph formatting properties of a text that is attached to the PowerPoint.TextRange .
PowerPoint.PlaceholderFormat	Represents the properties of a <code>placeholder</code> shape.
PowerPoint.Presentation	
PowerPoint.RequestContext	The RequestContext object facilitates requests to the PowerPoint application. Since the Office add-in and the PowerPoint application run in two different processes, the request context is required to get access to the PowerPoint object model from the add-in.
PowerPoint.Shape	Represents a single shape in the slide.
PowerPoint.ShapeCollection	Represents the collection of shapes.
PowerPoint.ShapeFill	Represents the fill formatting of a shape object.
PowerPoint.ShapeFont	Represents the font attributes, such as font name, font size, and color, for a shape's TextRange object.
PowerPoint.ShapeGroup	Represents a shape group inside a presentation. To get the corresponding Shape object, use <code>ShapeGroup.shape</code> .
PowerPoint.ShapeLineFormat	Represents the line formatting for the shape object. For images and geometric shapes, line formatting represents the border of the shape.
PowerPoint.ShapeScopedCollection	Represents a collection of shapes.
PowerPoint.Slide	Represents a single slide of a presentation.
PowerPoint.SlideBackground	Represents a background of a slide.
PowerPoint.SlideBackgroundFill	Represents the fill formatting of a slide background object.
PowerPoint.SlideBackgroundGradientFill	Represents PowerPoint.SlideBackground gradient fill properties.
PowerPoint.SlideBackgroundPatternFill	Represents PowerPoint.SlideBackground pattern fill properties.
PowerPoint.SlideBackgroundPictureOrTextureFill	Represents PowerPoint.SlideBackground picture or texture fill properties.

PowerPoint.SlideBackgroundSolidFill	Represents PowerPoint.SlideBackground solid fill properties.
PowerPoint.SlideCollection	Represents the collection of slides in the presentation.
PowerPoint.SlideLayout	Represents the layout of a slide.
PowerPoint.SlideLayoutBackground	Represents the background of a slide layout.
PowerPoint.SlideLayoutCollection	Represents the collection of layouts provided by the Slide Master for slides.
PowerPoint.SlideMaster	Represents the Slide Master of a slide.
PowerPoint.SlideMasterBackground	Represents the background of a slide master.
PowerPoint.SlideMasterCollection	Represents the collection of Slide Masters in the presentation.
PowerPoint.SlideScopedCollection	Represents a collection of slides in the presentation.
PowerPoint.Table	Represents a table.
PowerPoint.TableCell	Represents a table.
PowerPoint.TableCellCollection	Represents a collection of table cells.
PowerPoint.TableColumn	Represents a column in a table.
PowerPoint.TableColumnCollection	Represents a collection of table columns.
PowerPoint.TableRow	Represents a row in a table.
PowerPoint.TableRowCollection	Represents a collection of table rows.
PowerPoint.TableStyleOptions	Represents the available table style options.
PowerPoint.Tag	Represents a single tag in the slide.

PowerPoint.Tag Collection	Represents the collection of tags.
PowerPoint.Text Frame	Represents the text frame of a shape object.
PowerPoint.Text Range	Contains the text that is attached to a shape, in addition to properties and methods for manipulating the text.
PowerPoint.Theme ColorScheme	Represents a theme color scheme.

Interfaces

 Expand table

PowerPoint.Add SlideOptions	Represents the available options when adding a new slide.
PowerPoint.Border Properties	Represents the properties for a table cell border.
PowerPoint.Fill Properties	Represents the fill formatting of a table cell.
PowerPoint.Font Properties	Represents the font attributes, such as font name, size, and color.
PowerPoint.Insert SlideOptions	Represents the available options when inserting slides.
PowerPoint.Interfaces.Binding CollectionData	An interface describing the data returned by calling <code>bindingCollection.toJSON()</code> .
PowerPoint.Interfaces.Binding CollectionLoad Options	Represents the collection of all the binding objects that are part of the presentation.
PowerPoint.Interfaces.Binding CollectionUpdate Data	An interface for updating data on the <code>BindingCollection</code> object, for use in <code>bindingCollection.set({ ... })</code> .
PowerPoint.Interfaces.Binding Data	An interface describing the data returned by calling <code>binding.toJSON()</code> .

PowerPoint. Interfaces.Binding LoadOptions	Represents an Office.js binding that is defined in the presentation.
PowerPoint. Interfaces.Border Data	An interface describing the data returned by calling <code>border.toJSON()</code> .
PowerPoint. Interfaces.Border LoadOptions	Represents the properties for a table cell border.
PowerPoint. Interfaces.Borders Data	An interface describing the data returned by calling <code>borders.toJSON()</code> .
PowerPoint. Interfaces.Borders LoadOptions	Represents the borders for a table cell.
PowerPoint. Interfaces.Border UpdateData	An interface for updating data on the <code>Border</code> object, for use in <code>border.set({ ... })</code> .
PowerPoint. Interfaces.Bullet FormatData	An interface describing the data returned by calling <code>bulletFormat.toJSON()</code> .
PowerPoint. Interfaces.Bullet FormatLoad Options	Represents the bullet formatting properties of a text that is attached to the <code>PowerPoint.ParagraphFormat</code> .
PowerPoint. Interfaces.Bullet FormatUpdate Data	An interface for updating data on the <code>BulletFormat</code> object, for use in <code>bulletFormat.set({ ... })</code> .
PowerPoint. Interfaces. CollectionLoad Options	Provides ways to load properties of only a subset of members of a collection.
PowerPoint. Interfaces.Custom PropertyCollection Data	An interface describing the data returned by calling <code>customPropertyCollection.toJSON()</code> .
PowerPoint. Interfaces.Custom	A collection of custom properties.

PropertyCollection LoadOptions	
PowerPoint. Interfaces.Custom PropertyCollection UpdateData	An interface for updating data on the <code>CustomPropertyCollection</code> object, for use in <code>customPropertyCollection.set({ ... })</code> .
PowerPoint. Interfaces.Custom PropertyData	An interface describing the data returned by calling <code>customProperty.toJSON()</code> .
PowerPoint. Interfaces.Custom PropertyLoad Options	Represents a custom property.
PowerPoint. Interfaces.Custom PropertyUpdate Data	An interface for updating data on the <code>CustomProperty</code> object, for use in <code>customProperty.set({ ... })</code> .
PowerPoint. Interfaces.Custom XmlPartCollection Data	An interface describing the data returned by calling <code>customXmlPartCollection.toJSON()</code> .
PowerPoint. Interfaces.Custom XmlPartCollection LoadOptions	A collection of custom XML parts.
PowerPoint. Interfaces.Custom XmlPartCollection UpdateData	An interface for updating data on the <code>CustomXmlPartCollection</code> object, for use in <code>customXmlPartCollection.set({ ... })</code> .
PowerPoint. Interfaces.Custom XmlPartData	An interface describing the data returned by calling <code>customXmlPart.toJSON()</code> .
PowerPoint. Interfaces.Custom XmlPartLoad Options	Represents a custom XML part object.
PowerPoint. Interfaces.Custom XmlPartScoped CollectionData	An interface describing the data returned by calling <code>customXmlPartScopedCollection.toJSON()</code> .

PowerPoint. Interfaces.Custom XmlPartScoped CollectionLoad Options	A scoped collection of custom XML parts. A scoped collection is the result of some operation (such as filtering by namespace). A scoped collection cannot be scoped any further.
PowerPoint. Interfaces.Custom XmlPartScoped CollectionUpdate Data	An interface for updating data on the <code>CustomXmlPartScopedCollection</code> object, for use in <code>customXmlPartScopedCollection.set({ ... })</code> .
PowerPoint. Interfaces. Document PropertiesData	An interface describing the data returned by calling <code>documentProperties.toJSON()</code> .
PowerPoint. Interfaces. Document PropertiesLoad Options	Represents presentation properties.
PowerPoint. Interfaces. Document PropertiesUpdate Data	An interface for updating data on the <code>DocumentProperties</code> object, for use in <code>documentProperties.set({ ... })</code> .
PowerPoint. Interfaces. Hyperlink CollectionData	An interface describing the data returned by calling <code>hyperlinkCollection.toJSON()</code> .
PowerPoint. Interfaces. Hyperlink CollectionLoad Options	Represents a collection of hyperlinks.
PowerPoint. Interfaces. Hyperlink CollectionUpdate Data	An interface for updating data on the <code>HyperlinkCollection</code> object, for use in <code>hyperlinkCollection.set({ ... })</code> .
PowerPoint. Interfaces. HyperlinkData	An interface describing the data returned by calling <code>hyperlink.toJSON()</code> .

PowerPoint. Interfaces. HyperlinkLoad Options	Represents a single hyperlink.
PowerPoint. Interfaces. HyperlinkUpdate Data	An interface for updating data on the <code>Hyperlink</code> object, for use in <code>hyperlink.set({ ... })</code> .
PowerPoint. Interfaces.Margins Data	An interface describing the data returned by calling <code>margins.toJSON()</code> .
PowerPoint. Interfaces.Margins LoadOptions	Represents the margins of a table cell.
PowerPoint. Interfaces.Margins UpdateData	An interface for updating data on the <code>Margins</code> object, for use in <code>margins.set({ ... })</code> .
PowerPoint. Interfaces.Page SetupData	An interface describing the data returned by calling <code>pageSetup.toJSON()</code> .
PowerPoint. Interfaces.Page SetupLoadOptions	Represents the page setup information for the presentation.
PowerPoint. Interfaces.Page SetupUpdateData	An interface for updating data on the <code>PageSetup</code> object, for use in <code>pageSetup.set({ ... })</code> .
PowerPoint. Interfaces. ParagraphFormat Data	An interface describing the data returned by calling <code>paragraphFormat.toJSON()</code> .
PowerPoint. Interfaces. ParagraphFormat LoadOptions	Represents the paragraph formatting properties of a text that is attached to the PowerPoint.TextRange .
PowerPoint. Interfaces. ParagraphFormat UpdateData	An interface for updating data on the <code>ParagraphFormat</code> object, for use in <code>paragraphFormat.set({ ... })</code> .
PowerPoint. Interfaces.	An interface describing the data returned by calling <code>placeholderFormat.toJSON()</code> .

Placeholder FormatData	
PowerPoint. Interfaces. Placeholder FormatLoad Options	Represents the properties of a <code>placeholder</code> shape.
PowerPoint. Interfaces. PresentationData	An interface describing the data returned by calling <code>presentation.toJSON()</code> .
PowerPoint.Interfaces.PresentationLoadOptions	
PowerPoint. Interfaces.Shape CollectionData	An interface describing the data returned by calling <code>shapeCollection.toJSON()</code> .
PowerPoint. Interfaces.Shape CollectionLoad Options	Represents the collection of shapes.
PowerPoint. Interfaces.Shape CollectionUpdate Data	An interface for updating data on the <code>ShapeCollection</code> object, for use in <code>shapeCollection.set({ ... })</code> .
PowerPoint. Interfaces.Shape Data	An interface describing the data returned by calling <code>shape.toJSON()</code> .
PowerPoint. Interfaces.Shape FillData	An interface describing the data returned by calling <code>shapeFill.toJSON()</code> .
PowerPoint. Interfaces.Shape FillLoadOptions	Represents the fill formatting of a shape object.
PowerPoint. Interfaces.Shape FillUpdateData	An interface for updating data on the <code>ShapeFill</code> object, for use in <code>shapeFill.set({ ... })</code> .
PowerPoint. Interfaces.Shape FontData	An interface describing the data returned by calling <code>shapeFont.toJSON()</code> .
PowerPoint. Interfaces.Shape	Represents the font attributes, such as font name, font size, and color, for a shape's <code>TextRange</code> object.

FontLoadOptions	
PowerPoint. Interfaces.Shape FontUpdateData	An interface for updating data on the <code>ShapeFont</code> object, for use in <code>shapeFont.set({ ... })</code> .
PowerPoint. Interfaces.Shape GroupData	An interface describing the data returned by calling <code>shapeGroup.toJSON()</code> .
PowerPoint. Interfaces.Shape GroupLoad Options	Represents a shape group inside a presentation. To get the corresponding Shape object, use <code>ShapeGroup.shape</code> .
PowerPoint. Interfaces.Shape LineFormatData	An interface describing the data returned by calling <code>shapeLineFormat.toJSON()</code> .
PowerPoint. Interfaces.Shape LineFormatLoad Options	Represents the line formatting for the shape object. For images and geometric shapes, line formatting represents the border of the shape.
PowerPoint. Interfaces.Shape LineFormatUpdate Data	An interface for updating data on the <code>ShapeLineFormat</code> object, for use in <code>shapeLineFormat.set({ ... })</code> .
PowerPoint. Interfaces.Shape LoadOptions	Represents a single shape in the slide.
PowerPoint. Interfaces.Shape ScopedCollection Data	An interface describing the data returned by calling <code>shapeScopedCollection.toJSON()</code> .
PowerPoint. Interfaces.Shape ScopedCollection LoadOptions	Represents a collection of shapes.
PowerPoint. Interfaces.Shape ScopedCollection UpdateData	An interface for updating data on the <code>ShapeScopedCollection</code> object, for use in <code>shapeScopedCollection.set({ ... })</code> .
PowerPoint. Interfaces.Shape UpdateData	An interface for updating data on the <code>Shape</code> object, for use in <code>shape.set({ ... })</code> .

PowerPoint. Interfaces.Slide BackgroundData	An interface describing the data returned by calling <code>slideBackground.toJSON()</code> .
PowerPoint. Interfaces.Slide BackgroundFill Data	An interface describing the data returned by calling <code>slideBackgroundFill.toJSON()</code> .
PowerPoint. Interfaces.Slide BackgroundFill LoadOptions	Represents the fill formatting of a slide background object.
PowerPoint. Interfaces.Slide Background GradientFillData	An interface describing the data returned by calling <code>slideBackgroundGradientFill.toJSON()</code> .
PowerPoint. Interfaces.Slide Background GradientFillLoad Options	Represents PowerPoint.SlideBackground gradient fill properties.
PowerPoint. Interfaces.Slide Background GradientFillUpdate Data	An interface for updating data on the <code>SlideBackgroundGradientFill</code> object, for use in <code>slideBackgroundGradientFill.set({ ... })</code> .
PowerPoint. Interfaces.Slide BackgroundLoad Options	Represents a background of a slide.
PowerPoint. Interfaces.Slide Background PatternFillData	An interface describing the data returned by calling <code>slideBackgroundPatternFill.toJSON()</code> .
PowerPoint. Interfaces.Slide Background PatternFillLoad Options	Represents PowerPoint.SlideBackground pattern fill properties.
PowerPoint. Interfaces.Slide Background	An interface for updating data on the <code>SlideBackgroundPatternFill</code> object, for use in <code>slideBackgroundPatternFill.set({ ... })</code> .

PatternFillUpdate Data	
PowerPoint. Interfaces.Slide Background PictureOrTexture FillData	An interface describing the data returned by calling <code>slideBackgroundPictureOrTextureFill.toJSON()</code> .
PowerPoint. Interfaces.Slide Background PictureOrTexture FillLoadOptions	Represents PowerPoint.SlideBackground picture or texture fill properties.
PowerPoint. Interfaces.Slide Background PictureOrTexture FillUpdateData	An interface for updating data on the <code>SlideBackgroundPictureOrTextureFill</code> object, for use in <code>slideBackgroundPictureOrTextureFill.set({ ... })</code> .
PowerPoint. Interfaces.Slide BackgroundSolid FillData	An interface describing the data returned by calling <code>slideBackgroundSolidFill.toJSON()</code> .
PowerPoint. Interfaces.Slide BackgroundSolid FillLoadOptions	Represents PowerPoint.SlideBackground solid fill properties.
PowerPoint. Interfaces.Slide BackgroundSolid FillUpdateData	An interface for updating data on the <code>SlideBackgroundSolidFill</code> object, for use in <code>slideBackgroundSolidFill.set({ ... })</code> .
PowerPoint. Interfaces.Slide Background UpdateData	An interface for updating data on the <code>SlideBackground</code> object, for use in <code>slideBackground.set({ ... })</code> .
PowerPoint. Interfaces.Slide CollectionData	An interface describing the data returned by calling <code>slideCollection.toJSON()</code> .
PowerPoint. Interfaces.Slide CollectionLoad Options	Represents the collection of slides in the presentation.

PowerPoint. Interfaces.Slide CollectionUpdate Data	An interface for updating data on the <code>SlideCollection</code> object, for use in <code>slideCollection.set({ ... })</code> .
PowerPoint. Interfaces.Slide Data	An interface describing the data returned by calling <code>slide.toJSON()</code> .
PowerPoint. Interfaces.Slide LayoutBackground Data	An interface describing the data returned by calling <code>slideLayoutBackground.toJSON()</code> .
PowerPoint. Interfaces.Slide LayoutBackground LoadOptions	Represents the background of a slide layout.
PowerPoint. Interfaces.Slide LayoutBackground UpdateData	An interface for updating data on the <code>SlideLayoutBackground</code> object, for use in <code>slideLayoutBackground.set({ ... })</code> .
PowerPoint. Interfaces.Slide LayoutCollection Data	An interface describing the data returned by calling <code>slideLayoutCollection.toJSON()</code> .
PowerPoint. Interfaces.Slide LayoutCollection LoadOptions	Represents the collection of layouts provided by the Slide Master for slides.
PowerPoint. Interfaces.Slide LayoutCollection UpdateData	An interface for updating data on the <code>SlideLayoutCollection</code> object, for use in <code>slideLayoutCollection.set({ ... })</code> .
PowerPoint. Interfaces.Slide LayoutData	An interface describing the data returned by calling <code>slideLayout.toJSON()</code> .
PowerPoint. Interfaces.Slide LayoutLoad Options	Represents the layout of a slide.
PowerPoint. Interfaces.Slide LoadOptions	Represents a single slide of a presentation.

PowerPoint. Interfaces.Slide Master BackgroundData	An interface describing the data returned by calling <code>slideMasterBackground.toJSON()</code> .
PowerPoint. Interfaces.Slide Master BackgroundLoad Options	Represents the background of a slide master.
PowerPoint. Interfaces.Slide MasterCollection Data	An interface describing the data returned by calling <code>slideMasterCollection.toJSON()</code> .
PowerPoint. Interfaces.Slide MasterCollection LoadOptions	Represents the collection of Slide Masters in the presentation.
PowerPoint. Interfaces.Slide MasterCollection UpdateData	An interface for updating data on the <code>SlideMasterCollection</code> object, for use in <code>slideMasterCollection.set({ ... })</code> .
PowerPoint. Interfaces.Slide MasterData	An interface describing the data returned by calling <code>slideMaster.toJSON()</code> .
PowerPoint. Interfaces.Slide MasterLoad Options	Represents the Slide Master of a slide.
PowerPoint. Interfaces.Slide ScopedCollection Data	An interface describing the data returned by calling <code>slideScopedCollection.toJSON()</code> .
PowerPoint. Interfaces.Slide ScopedCollection LoadOptions	Represents a collection of slides in the presentation.
PowerPoint. Interfaces.Slide ScopedCollection UpdateData	An interface for updating data on the <code>SlideScopedCollection</code> object, for use in <code>slideScopedCollection.set({ ... })</code> .

PowerPoint. Interfaces.Table CellCollectionData	An interface describing the data returned by calling <code>tableCellCollection.toJSON()</code> .
PowerPoint. Interfaces.Table CellCollectionLoad Options	Represents a collection of table cells.
PowerPoint. Interfaces.Table CellCollection UpdateData	An interface for updating data on the <code>TableCellCollection</code> object, for use in <code>tableCellCollection.set({ ... })</code> .
PowerPoint. Interfaces.Table CellData	An interface describing the data returned by calling <code>tableCell.toJSON()</code> .
PowerPoint. Interfaces.Table CellLoadOptions	Represents a table.
PowerPoint. Interfaces.Table CellUpdateData	An interface for updating data on the <code>TableCell</code> object, for use in <code>tableCell.set({ ... })</code> .
PowerPoint. Interfaces.Table ColumnCollection Data	An interface describing the data returned by calling <code>tableColumnCollection.toJSON()</code> .
PowerPoint. Interfaces.Table ColumnCollection LoadOptions	Represents a collection of table columns.
PowerPoint. Interfaces.Table ColumnCollection UpdateData	An interface for updating data on the <code>TableColumnCollection</code> object, for use in <code>tableColumnCollection.set({ ... })</code> .
PowerPoint. Interfaces.Table ColumnData	An interface describing the data returned by calling <code>tableColumn.toJSON()</code> .
PowerPoint. Interfaces.Table ColumnLoad Options	Represents a column in a table.


PowerPoint. Interfaces.Table ColumnUpdate Data	An interface for updating data on the <code>TableColumn</code> object, for use in <code>tableColumn.set({ ... })</code> .
PowerPoint. Interfaces.Table Data	An interface describing the data returned by calling <code>table.toJSON()</code> .
PowerPoint. Interfaces.Table LoadOptions	Represents a table.
PowerPoint. Interfaces.Table RowCollection Data	An interface describing the data returned by calling <code>tableRowCollection.toJSON()</code> .
PowerPoint. Interfaces.Table RowCollection LoadOptions	Represents a collection of table rows.
PowerPoint. Interfaces.Table RowCollection UpdateData	An interface for updating data on the <code>TableRowCollection</code> object, for use in <code>tableRowCollection.set({ ... })</code> .
PowerPoint. Interfaces.Table RowData	An interface describing the data returned by calling <code>tableRow.toJSON()</code> .
PowerPoint. Interfaces.Table RowLoadOptions	Represents a row in a table.
PowerPoint. Interfaces.Table RowUpdateData	An interface for updating data on the <code>TableRow</code> object, for use in <code>tableRow.set({ ... })</code> .
PowerPoint. Interfaces.Table StyleOptionsData	An interface describing the data returned by calling <code>tableStyleOptions.toJSON()</code> .
PowerPoint. Interfaces.Table StyleOptionsLoad Options	Represents the available table style options.
PowerPoint. Interfaces.Table	An interface for updating data on the <code>TableStyleOptions</code> object, for use in <code>tableStyleOptions.set({ ... })</code> .

StyleOptions UpdateData	
PowerPoint. Interfaces.Tag CollectionData	An interface describing the data returned by calling <code>tagCollection.toJSON()</code> .
PowerPoint. Interfaces.Tag CollectionLoad Options	Represents the collection of tags.
PowerPoint. Interfaces.Tag CollectionUpdate Data	An interface for updating data on the <code>TagCollection</code> object, for use in <code>tagCollection.set({ ... })</code> .
PowerPoint. Interfaces.TagData	An interface describing the data returned by calling <code>tag.toJSON()</code> .
PowerPoint. Interfaces.TagLoad Options	Represents a single tag in the slide.
PowerPoint. Interfaces.Tag UpdateData	An interface for updating data on the <code>Tag</code> object, for use in <code>tag.set({ ... })</code> .
PowerPoint. Interfaces.Text FrameData	An interface describing the data returned by calling <code>textFrame.toJSON()</code> .
PowerPoint. Interfaces.Text FrameLoad Options	Represents the text frame of a shape object.
PowerPoint. Interfaces.Text FrameUpdateData	An interface for updating data on the <code>TextFrame</code> object, for use in <code>textFrame.set({ ... })</code> .
PowerPoint. Interfaces.Text RangeData	An interface describing the data returned by calling <code>textRange.toJSON()</code> .
PowerPoint. Interfaces.Text RangeLoad Options	Contains the text that is attached to a shape, in addition to properties and methods for manipulating the text.

PowerPoint.Interfaces.TextRangeUpdateData	An interface for updating data on the <code>TextRange</code> object, for use in <code>textRange.set({ ... })</code> .
PowerPoint.ShapeAddOptions	Represents the available options when adding shapes.
PowerPoint.ShapeGetImageOptions	Represents the available options when getting an image of a shape. The image is scaled to fit into the desired dimensions. If width and height aren't specified, the true size of the shape is used. If only one of either width or height is specified, the other will be calculated to preserve aspect ratio. The resulting dimensions will automatically be clamped to the maximum supported size if too large.
PowerPoint.SlideBackgroundGradientFillOptions	Represents the available options for setting a PowerPoint.SlideBackground gradient fill.
PowerPoint.SlideBackgroundPatternFillOptions	Represents the available options for setting a PowerPoint.SlideBackground pattern fill.
PowerPoint.SlideBackgroundPictureOrTextureFillOptions	Represents PowerPoint.SlideBackground picture or texture fill options.
PowerPoint.SlideBackgroundSolidFillOptions	Represents the available options for setting a PowerPoint.SlideBackground solid fill.
PowerPoint.SlideGetImageOptions	Represents the available options when getting an image of a slide.
PowerPoint.TableAddOptions	Represents the available options when adding a table.
PowerPoint.TableCellBorders	Represents the borders of a table cell.
PowerPoint.TableCellMargins	Represents the margins of a table cell.
PowerPoint.TableCellProperties	Represents the table cell properties to update.
PowerPoint.TableClearOptions	Represents the available options when clearing a table.
PowerPoint.TableColumnProperties	Provides the table column properties.

PowerPoint.TableMergedAreaProperties	Represents the properties of a merged area of cells in a table.
PowerPoint.TableRowProperties	Provides the table row properties.
PowerPoint.TextRun	Represents a sequence of one or more characters with the same font attributes.


Enums

 Expand table

PowerPoint.BindingType	Represents the possible binding types.
PowerPoint.BulletStyle	Specifies the style of a bullet.
PowerPoint.BulletType	Specifies the type of a bullet.
PowerPoint.ConnectorType	Specifies the connector type for line shapes.
PowerPoint.DocumentPropertyType	Specifies the document property type for custom properties.
PowerPoint.ErrorCodes	
PowerPoint.GeometricShapeType	Specifies the shape type for a <code>GeometricShape</code> object.
PowerPoint.InsertSlideFormatting	Specifies the formatting options for when slides are inserted.
PowerPoint.ParagraphHorizontalAlignment	Represents the horizontal alignment of the PowerPoint.TextFrame in a PowerPoint.Shape .
PowerPoint.PlaceholderType	Specifies the type of a placeholder.

PowerPoint.ShapeAutoSize	Determines the type of automatic sizing allowed.
PowerPoint.ShapeFillType	Specifies a shape's fill type.
PowerPoint.ShapeFontUnderlineStyle	The type of underline applied to a font.
PowerPoint.ShapeGetImageFormatType	Represents the format of an image.
PowerPoint.ShapeLineDashStyle	Specifies the dash style for a line.
PowerPoint.ShapeLineStyle	Specifies the style for a line.
PowerPoint.ShapeType	Specifies the type of a shape.
PowerPoint.ShapeZOrder	Use with <code>setZOrder</code> to move the specified shape up or down the collection's z-order, which shifts it in front of or behind other shapes.
PowerPoint.SlideBackgroundFillType	Specifies the fill type for a PowerPoint.SlideBackground .
PowerPoint.SlideBackgroundGradientFillType	Specifies the gradient fill type for a PowerPoint.SlideBackgroundGradientFill .
PowerPoint.SlideBackgroundPatternFillType	Specifies the pattern fill type for a PowerPoint.SlideBackgroundPatternFill .
PowerPoint.SlideLayoutType	Specifies the type of a slide layout.
PowerPoint.TableStyle	Represents the available built-in table styles.
PowerPoint.TextVerticalAlignment	Represents the vertical alignment of a PowerPoint.TextFrame in a PowerPoint.Shape . If one of the centered options is selected, the contents of the <code>TextFrame</code> will be centered horizontally within the <code>Shape</code> as a group. To change the horizontal alignment of a text, see PowerPoint.ParagraphFormat and PowerPoint.ParagraphHorizontalAlignment .
PowerPoint.	Specifies the theme colors used in PowerPoint.

Functions

 Expand table

PowerPoint.createPresentation(base64File)	Creates and opens a new presentation. Optionally, the presentation can be prepopulated with a Base64-encoded .pptx file. [API set: PowerPointApi 1.1]
PowerPoint.run(batch)	Executes a batch script that performs actions on the PowerPoint object model, using a new RequestContext. When the promise is resolved, any tracked objects that were automatically allocated during execution will be released.
PowerPoint.run(object, batch)	Executes a batch script that performs actions on the PowerPoint object model, using the RequestContext of a previously-created API object. When the promise is resolved, any tracked objects that were automatically allocated during execution will be released.
PowerPoint.run(objects, batch)	Executes a batch script that performs actions on the PowerPoint object model, using the RequestContext of previously-created API objects.

Function Details

PowerPoint.createPresentation(base64File)

Creates and opens a new presentation. Optionally, the presentation can be prepopulated with a Base64-encoded .pptx file.

[[API set: PowerPointApi 1.1](#)]

TypeScript

```
export function createPresentation(base64File?: string): Promise<void>;
```

Parameters

base64File string

Optional. The Base64-encoded .pptx file. The default value is null. The maximum length of the string is 71,680,000 characters.

Returns

Promise<void>

Examples

TypeScript

```
const myFile = <HTMLInputElement>document.getElementById("file");
const reader = new FileReader();

reader.onload = (event) => {
    // Remove the metadata before the base64-encoded string.
    const startIndex = reader.result.toString().indexOf("base64,");
    const 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]);
```

PowerPoint.run(batch)

Executes a batch script that performs actions on the PowerPoint object model, using a new RequestContext. When the promise is resolved, any tracked objects that were automatically allocated during execution will be released.

TypeScript

```
export function run<T>(batch: (context: PowerPoint.RequestContext) =>
OfficeExtension.IPromise<T>): OfficeExtension.IPromise<T>;
```

Parameters

batch (context: [PowerPoint.RequestContext](#)) => [OfficeExtension.IPromise](#)<T>

A function that takes in a RequestContext and returns a promise (typically, just the result of "context.sync()"). The context parameter facilitates requests to the PowerPoint application. Since the Office add-in and the PowerPoint application run in two different processes, the RequestContext is required to get access to the PowerPoint object model from the add-in.

Returns

[OfficeExtension.IPromise](#)<T>

PowerPoint.run(object, batch)

Executes a batch script that performs actions on the PowerPoint object model, using the RequestContext of a previously-created API object. When the promise is resolved, any tracked objects that were automatically allocated during execution will be released.

TypeScript

```
export function run<T>(object: OfficeExtension.ClientObject, batch: (context: PowerPoint.RequestContext) => OfficeExtension.IPromise<T>): OfficeExtension.IPromise<T>;
```

Parameters

object [OfficeExtension.ClientObject](#)

A previously-created API object. The batch will use the same RequestContext as the passed-in object, which means that any changes applied to the object will be picked up by "context.sync()".

batch (context: [PowerPoint.RequestContext](#)) => [OfficeExtension.IPromise<T>](#)

A function that takes in a RequestContext and returns a promise (typically, just the result of "context.sync()"). The context parameter facilitates requests to the PowerPoint application. Since the Office add-in and the PowerPoint application run in two different processes, the RequestContext is required to get access to the PowerPoint object model from the add-in.

Returns

[OfficeExtension.IPromise<T>](#)

PowerPoint.run(objects, batch)

Executes a batch script that performs actions on the PowerPoint object model, using the RequestContext of previously-created API objects.

TypeScript

```
export function run<T>(objects: OfficeExtension.ClientObject[], batch: (context: PowerPoint.RequestContext) => OfficeExtension.IPromise<T>): OfficeExtension.IPromise<T>;
```

Parameters

objects [OfficeExtension.ClientObject\[\]](#)

An array of previously-created API objects. The array will be validated to make sure that all of the objects share the same context. The batch will use this shared RequestContext, which means that any changes applied to these objects will be picked up by "context.sync()".

batch (context: [PowerPoint.RequestContext](#)) => [OfficeExtension.IPromise<T>](#)

A function that takes in a RequestContext and returns a promise (typically, just the result of "context.sync()"). The context parameter facilitates requests to the PowerPoint application. Since the Office add-in and the PowerPoint application run in two different processes, the RequestContext is required to get access to the PowerPoint object model from the add-in.

Returns

[OfficeExtension.IPromise<T>](#)

PowerPoint JavaScript object model in Office Add-ins

06/20/2025

This article describes concepts that are fundamental to using the [PowerPoint JavaScript API](#) to build add-ins.

Office.js APIs for PowerPoint

A PowerPoint add-in interacts with objects in PowerPoint by using the Office JavaScript API. This includes two JavaScript object models:

- **PowerPoint JavaScript API:** The [PowerPoint JavaScript API](#) provides strongly-typed objects that work with the presentation, slides, tables, shapes, formatting, and more. To learn about the asynchronous nature of the PowerPoint APIs and how they work with the presentation, see [Using the application-specific API model](#).
- **Common APIs:** The [Common API](#) give access to features such as UI, dialogs, and client settings that are common across multiple Office applications. To learn more about using the Common API, see [Common JavaScript API object model](#).

While you'll likely use the PowerPoint JavaScript API to develop the majority of functionality in add-ins that target PowerPoint, you'll also use objects in the Common API. For example:

- **Office.Context:** The `Office.Context` object represents the runtime environment of the add-in and provides access to key objects of the API. It consists of presentation configuration details such as `contentLanguage` and `officeTheme` and also provides information about the add-in's runtime environment such as `host` and `platform`. Additionally, it provides the `requirements.isSetSupported()` method, which you can use to check whether a specified requirement set is supported by the PowerPoint application where the add-in is running.
- **Office.Document:** The `Office.Document` object provides the `getFileAsync()` method, which you can use to download the PowerPoint file where the add-in is running. It also provides the `getActiveViewAsync()` method, which you can use to check whether the presentation is in a "read" or "edit" view. "edit" corresponds to any of the views in which you can edit slides: Normal, Slide Sorter, or Outline View. "read" corresponds to either Slide Show or Reading View.


PowerPoint-specific object model

To understand the PowerPoint APIs, you must understand how key components of a presentation are related to one another.

- The presentation contains slides and presentation-level entities such as settings and custom XML parts.
- A slide contains content like shapes, text, and tables.
- A layout determines how a slide's content is organized and displayed.

For the full set of objects supported by the PowerPoint JavaScript API, see [PowerPoint JavaScript API](#).

See also

- [PowerPoint JavaScript API overview](#)
- [Build your first PowerPoint add-in](#)
- [PowerPoint add-in tutorial](#)
- [PowerPoint JavaScript API reference](#)
- [Learn about the Microsoft 365 Developer Program](#) 

Add and delete slides in PowerPoint

Article • 07/21/2022

A PowerPoint add-in can add slides to the presentation and optionally specify which slide master, and which layout of the master, is used for the new slide. The add-in can also delete slides.

The APIs for adding slides are primarily used in scenarios where the IDs of the slide masters and layouts in the presentation are known at coding time or can be found in a data source at runtime. In such a scenario, either you or the customer must create and maintain a data source that correlates the selection criterion (such as the names or images of slide masters and layouts) with the IDs of the slide masters and layouts. The APIs can also be used in scenarios where the user can insert slides that use the default slide master and the master's default layout, and in scenarios where the user can select an existing slide and create a new one with the same slide master and layout (but not the same content). See [Selecting which slide master and layout to use](#) for more information about this.

Add a slide with SlideCollection.add

Add slides with the [SlideCollection.add](#) method. The following is a simple example in which a slide that uses the presentation's default slide master and the first layout of that master is added. The method always adds new slides to the end of the presentation. The following is an example.

JavaScript

```
async function addSlide() {
    await PowerPoint.run(async function(context) {
        context.presentation.slides.add();

        await context.sync();
    });
}
```

Select which slide master and layout to use

Use the [AddSlideOptions](#) parameter to control which slide master is used for the new slide and which layout within the master is used. The following is an example. About this code, note:

- You can include either or both the properties of the `AddSlideOptions` object.
- If both properties are used, then the specified layout must belong to the specified master or an error is thrown.
- If the `masterId` property isn't present (or its value is an empty string), then the default slide master is used and the `layoutId` must be a layout of that slide master.
- The default slide master is the slide master used by the last slide in the presentation. (In the unusual case where there are currently no slides in the presentation, then the default slide master is the first slide master in the presentation.)
- If the `layoutId` property isn't present (or its value is an empty string), then the first layout of the master that is specified by the `masterId` is used.
- Both properties are strings of one of three possible forms: *nnnnnnnnnn#*, *#mmmmmmmmmm*, or *nnnnnnnnnn#mmmmmmmmmm*, where *nnnnnnnnnn* is the master's or layout's ID (typically 10 digits) and *mmmmmmmmmm* is the master's or layout's creation ID (typically 6 - 10 digits). Some examples are `2147483690#2908289500`, `2147483690#`, and `#2908289500`.

JavaScript

```
async function addSlide() {
    await PowerPoint.run(async function(context) {
        context.presentation.slides.add({
            slideMasterId: "2147483690#2908289500",
            layoutId: "2147483691#2499880"
        });

        await context.sync();
    });
}
```

There is no practical way that users can discover the ID or creation ID of a slide master or layout. For this reason, you can really only use the `AddSlideOptions` parameter when either you know the IDs at coding time or your add-in can discover them at runtime. Because users can't be expected to memorize the IDs, you also need a way to enable the user to select slides, perhaps by name or by an image, and then correlate each title or image with the slide's ID.

Accordingly, the `AddSlideOptions` parameter is primarily used in scenarios in which the add-in is designed to work with a specific set of slide masters and layouts whose IDs are known. In such a scenario, either you or the customer must create and maintain a data source that correlates a selection criterion (such as slide master and layout names or images) with the corresponding IDs or creation IDs.

Have the user choose a matching slide

If your add-in can be used in scenarios where the new slide should use the same combination of slide master and layout that is used by an *existing* slide, then your add-in can (1) prompt the user to select a slide and (2) read the IDs of the slide master and layout. The following steps show how to read the IDs and add a slide with a matching master and layout.

1. Create a function to get the index of the selected slide. The following is an example. About this code, note:

- It uses the [Office.context.document.getSelectedDataAsync](#) method of the Common JavaScript APIs.
- The call to `getSelectedDataAsync` is embedded in a Promise-returning function. For more information about why and how to do this, see [Wrap Common APIs in promise-returning functions](#).
- `getSelectedDataAsync` returns an array because multiple slides can be selected. In this scenario, the user has selected just one, so the code gets the first (0th) slide, which is the only one selected.
- The `index` value of the slide is the 1-based value the user sees beside the slide in the thumbnails pane.

JavaScript

```
function getSelectedSlideIndex() {
    return new OfficeExtension.Promise<number>(function(resolve,
reject) {

Office.context.document.getSelectedDataAsync(Office.CoercionType.SlideRange, function(asyncResult) {
    try {
        if (asyncResult.status ===
Office.AsyncResultStatus.Failed) {
            reject(console.error(asyncResult.error.message));
        } else {
            resolve(asyncResult.value.slides[0].index);
        }
    }
    catch (error) {
        reject(console.log(error));
    }
});
});
}
```

2. Call your new function inside the [PowerPoint.run\(\)](#) of the main function that adds the slide. The following is an example.

JavaScript

```
async function addSlideWithMatchingLayout() {
  await PowerPoint.run(async function(context) {

    let selectedIndex = await getSelectedSlideIndex();

    // Decrement the index because the value returned by
    getSelectedSlideIndex()
    // is 1-based, but SlideCollection.getItemAt() is 0-based.
    const realSlideIndex = selectedIndex - 1;
    const selectedSlide =
context.presentation.slides.getItemAt(realSlideIndex).load("slideMaster
/id, layout/id");

    await context.sync();

    context.presentation.slides.add({
      slideMasterId: selectedSlide.slideMaster.id,
      layoutId: selectedSlide.layout.id
    });

    await context.sync();
  });
}
```

Delete slides

Delete a slide by getting a reference to the [Slide](#) object that represents the slide and call the `Slide.delete` method. The following is an example in which the 4th slide is deleted.

JavaScript

```
async function deleteSlide() {
  await PowerPoint.run(async function(context) {

    // The slide index is zero-based.
    const slide = context.presentation.slides.getItemAt(3);
    slide.delete();

    await context.sync();
  });
}
```

Insert slides in a PowerPoint presentation

Article • 09/20/2022


A PowerPoint add-in can insert slides from one presentation into the current presentation by using PowerPoint's application-specific JavaScript library. You can control whether the inserted slides keep the formatting of the source presentation or the formatting of the target presentation.

The slide insertion APIs are primarily used in presentation template scenarios: There are a small number of known presentations which serve as pools of slides that can be inserted by the add-in. In such a scenario, either you or the customer must create and maintain a data source that correlates the selection criterion (such as slide titles or images) with slide IDs. The APIs can also be used in scenarios where the user can insert slides from any arbitrary presentation, but in that scenario the user is effectively limited to inserting *all* the slides from the source presentation. See [Selecting which slides to insert](#) for more information about this.

There are two steps to inserting slides from one presentation into another.

1. Convert the source presentation file (.pptx) into a base64-formatted string.
2. Use the `insertSlidesFromBase64` method to insert one or more slides from the base64 file into the current presentation.

Convert the source presentation to base64

There are many ways to convert a file to base64. Which programming language and library you use, and whether to convert on the server-side of your add-in or the client-side is determined by your scenario. Most commonly, you'll do the conversion in JavaScript on the client-side by using a [FileReader](#)  object. The following example shows this practice.

1. Begin by getting a reference to the source PowerPoint file. In this example, we will use an `<input>` control of type `file` to prompt the user to choose a file. Add the following markup to the add-in page.

HTML

```
<section>
  <p>Select a PowerPoint presentation from which to insert slides</p>
  <form>
```

```
<input type="file" id="file" />
</form>
</section>
```

This markup adds the UI in the following screenshot to the page.

Select a PowerPoint presentation from which to insert slides

Choose File No file chosen

ⓘ Note

There are many other ways to get a PowerPoint file. For example, if the file is stored on OneDrive or SharePoint, you can use Microsoft Graph to download it. For more information, see [Working with files in Microsoft Graph](#) and [Access Files with Microsoft Graph](#).

2. Add the following code to the add-in's JavaScript to assign a function to the input control's `change` event. (You create the `storeFileAsBase64` function in the next step.)

JavaScript

```
$("#file").on("change", storeFileAsBase64);
```

3. Add the following code. Note the following about this code.

- The `reader.readAsDataURL` method converts the file to base64 and stores it in the `reader.result` property. When the method completes, it triggers the `onload` event handler.
- The `onload` event handler trims metadata off of the encoded file and stores the encoded string in a global variable.
- The base64-encoded string is stored globally because it will be read by another function that you create in a later step.

JavaScript

```
let chosenFileBase64;

async function storeFileAsBase64() {
  const reader = new FileReader();
```

```

reader.onload = async (event) => {
    const startIndex = reader.result.toString().indexOf("base64,");
    const copyBase64 = reader.result.toString().substr(startIndex +
7);

    chosenFileBase64 = copyBase64;
};

const myFile = document.getElementById("file") as HTMLInputElement;
reader.readAsDataURL(myFile.files[0]);
}

```

Insert slides with insertSlidesFromBase64

Your add-in inserts slides from another PowerPoint presentation into the current presentation with the [Presentation.insertSlidesFromBase64](#) method. The following is a simple example in which all of the slides from the source presentation are inserted at the beginning of the current presentation and the inserted slides keep the formatting of the source file. Note that `chosenFileBase64` is a global variable that holds a base64-encoded version of a PowerPoint presentation file.

JavaScript

```

async function insertAllSlides() {
    await PowerPoint.run(async function(context) {
        context.presentation.insertSlidesFromBase64(chosenFileBase64);
        await context.sync();
    });
}

```

You can control some aspects of the insertion result, including where the slides are inserted and whether they get the source or target formatting , by passing an [InsertSlideOptions](#) object as a second parameter to `insertSlidesFromBase64`. The following is an example. About this code, note:

- There are two possible values for the `formatting` property: "UseDestinationTheme" and "KeepSourceFormatting". Optionally, you can use the `InsertSlideFormatting` enum, (e.g., `PowerPoint.InsertSlideFormatting.useDestinationTheme`).
- The function will insert the slides from the source presentation immediately after the slide specified by the `targetSlideId` property. The value of this property is a string of one of three possible forms: *nnn#*, *#mmmmmmmmmm*, or *nnn#mmmmmmmmmm*, where *nnn* is the slide's ID (typically 3 digits) and *mmmmmmmmmm* is the slide's creation ID (typically 9 digits). Some examples are `267#763315295`, `267#`, and `#763315295`.

```

async function insertSlidesDestinationFormatting() {
  await PowerPoint.run(async function(context) {
    context.presentation
      .insertSlidesFromBase64(chosenFileBase64,
        {
          formatting: "UseDestinationTheme",
          targetSlideId: "267#"
        }
      );
    await context.sync();
  });
}

```

Of course, you typically won't know at coding time the ID or creation ID of the target slide. More commonly, an add-in will ask users to select the target slide. The following steps show how to get the *nnn#* ID of the currently selected slide and use it as the target slide.

1. Create a function that gets the ID of the currently selected slide by using the [Office.context.document.getSelectedDataAsync](#) method of the Common JavaScript APIs. The following is an example. Note that the call to `getSelectedDataAsync` is embedded in a Promise-returning function. For more information about why and how to do this, see [Wrap Common-APIs in promise-returning functions](#).

```

function getSelectedSlideID() {
  return new OfficeExtension.Promise<string>(function (resolve, reject) {
    Office.context.document.getSelectedDataAsync(Office.CoercionType.SlideRange, function (asyncResult) {
      try {
        if (asyncResult.status === Office.AsyncResultStatus.Failed) {
          reject(console.error(asyncResult.error.message));
        } else {
          resolve(asyncResult.value.slides[0].id);
        }
      }
      catch (error) {
        reject(console.log(error));
      }
    });
  });
}

```


2. Call your new function inside the `PowerPoint.run()` of the main function and pass the ID that it returns (concatenated with the "#" symbol) as the value of the `targetSlideId` property of the `InsertSlideOptions` parameter. The following is an example.

JavaScript

```
async function insertAfterSelectedSlide() {
  await PowerPoint.run(async function(context) {

    const selectedSlideID = await getSelectedSlideID();

    context.presentation.insertSlidesFromBase64(chosenFileBase64, {
      formatting: "UseDestinationTheme",
      targetSlideId: selectedSlideID + "#"
    });

    await context.sync();
  });
}
```

Selecting which slides to insert

You can also use the `InsertSlideOptions` parameter to control which slides from the source presentation are inserted. You do this by assigning an array of the source presentation's slide IDs to the `sourceSlideIds` property. The following is an example that inserts four slides. Note that each string in the array must follow one or another of the patterns used for the `targetSlideId` property.

JavaScript

```
async function insertAfterSelectedSlide() {
  await PowerPoint.run(async function(context) {
    const selectedSlideID = await getSelectedSlideID();
    context.presentation.insertSlidesFromBase64(chosenFileBase64, {
      formatting: "UseDestinationTheme",
      targetSlideId: selectedSlideID + "#",
      sourceSlideIds: ["267#763315295", "256#", "#926310875", "1270#"]
    });

    await context.sync();
  });
}
```

❗ Note

The slides will be inserted in the same relative order in which they appear in the source presentation, regardless of the order in which they appear in the array.

There is no practical way that users can discover the ID or creation ID of a slide in the source presentation. For this reason, you can really only use the `sourceSlideIds` property when either you know the source IDs at coding time or your add-in can retrieve them at runtime from some data source. Because users cannot be expected to memorize slide IDs, you also need a way to enable the user to select slides, perhaps by title or by an image, and then correlate each title or image with the slide's ID.

Accordingly, the `sourceSlideIds` property is primarily used in presentation template scenarios: The add-in is designed to work with a specific set of presentations that serve as pools of slides that can be inserted. In such a scenario, either you or the customer must create and maintain a data source that correlates a selection criterion (such as titles or images) with slide IDs or slide creation IDs that has been constructed from the set of possible source presentations.

Get the whole document from an add-in for PowerPoint or Word

Article • 02/12/2025

You can create an Office Add-in to send or publish a PowerPoint presentation or Word document to a remote location. This article demonstrates how to build a simple task pane add-in for PowerPoint or Word that gets all of the presentation or document as a data object and sends that data to a web server via an HTTP request.

Prerequisites for creating an add-in for PowerPoint or Word

This article assumes that you are using a text editor to create the task pane add-in for PowerPoint or Word. To create the task pane add-in, you must create the following files.

- On a shared network folder or on a web server, you need the following files.
 - An HTML file (**GetDoc_App.html**) that contains the user interface plus links to the JavaScript files (including Office.js and application-specific .js files) and Cascading Style Sheet (CSS) files.
 - A JavaScript file (**GetDoc_App.js**) to contain the programming logic of the add-in.
 - A CSS file (**Program.css**) to contain the styles and formatting for the add-in.
- A manifest file (**GetDoc_App.xml** or **GetDoc_App.json**) for the add-in, available on a shared network folder or add-in catalog. The manifest file must point to the location of the HTML file mentioned previously.

Alternatively, you can create an add-in for your Office application using one of the following options. You won't have to create new files as the equivalent of each required file will be available for you to update. For example, the Yeoman generator options include `./src/taskpane/taskpane.html`, `./src/taskpane/taskpane.js`, `./src/taskpane/taskpane.css`, and `./manifest.xml`.

- PowerPoint
 - [Visual Studio](#)
 - [Yeoman generator for Office Add-ins](#)
- Word
 - [Visual Studio](#)

- [Yeoman generator for Office Add-ins](#)

Core concepts to know for creating a task pane add-in

Before you begin creating this add-in for PowerPoint or Word, you should be familiar with building Office Add-ins and working with HTTP requests. This article doesn't discuss how to decode Base64-encoded text from an HTTP request on a web server.

Create the manifest for the add-in

The manifest file for an Office Add-in provides important information about the add-in: what applications can host it, the location of the HTML file, the add-in title and description, and many other characteristics.

In a text editor, add the following code to the manifest file. If you're using a Visual Studio project, select the "Add-in only manifest" option.

Unified manifest for Microsoft 365

ⓘ Note

The unified manifest is generally available for production Outlook add-ins. It's available only for preview in Excel, PowerPoint, and Word add-ins.

JSON

```
{
  "$schema": "https://developer.microsoft.com/json-
schemas/teams/vDevPreview/MicrosoftTeams.schema.json#",
  "manifestVersion": "devPreview",
  "version": "1.0.0.0",
  "id": "[Replace_With_Your_GUID]",
  "localizationInfo": {
    "defaultLanguageTag": "en-us"
  },
  "developer": {
    "name": "[Provider Name e.g., Contoso]",
    "websiteUrl": "[Insert the URL for the app e.g.,
https://www.contoso.com]",
    "privacyUrl": "[Insert the URL of a page that provides privacy
information for the app e.g., https://www.contoso.com/privacy]",
    "termsOfUseUrl": "[Insert the URL of a page that provides terms
of use for the app e.g., https://www.contoso.com/servicesagreement]"
  },
  "name": {
```

```

        "short": "Get Doc add-in",
        "full": "Get Doc add-in"
    },
    "description": {
        "short": "My get PowerPoint or Word document add-in.",
        "full": "My get PowerPoint or Word document add-in."
    },
    "icons": {
        "outline": "_layouts/images/general/office_logo.jpg",
        "color": "_layouts/images/general/office_logo.jpg"
    },
    "accentColor": "#230201",
    "validDomains": [
        "https://www.contoso.com"
    ],
    "showLoadingIndicator": false,
    "isFullScreen": false,
    "defaultBlockUntilAdminAction": false,
    "authorization": {
        "permissions": {
            "resourceSpecific": [
                {
                    "name": "Document.ReadWrite.User",
                    "type": "Delegated"
                }
            ]
        }
    },
    "extensions": [
        {
            "requirements": {
                "scopes": [
                    "document",
                    "presentation"
                ]
            },
            "alternates": [
                {
                    "alternateIcons": {
                        "icon": {
                            "size": 32,
                            "url":
"http://officeimg.vo.msecnd.net/_layouts/images/general/office_logo.jpg"
                        },
                        "highResolutionIcon": {
                            "size": 64,
                            "url":
"http://officeimg.vo.msecnd.net/_layouts/images/general/office_logo.jpg"
                        }
                    }
                }
            ]
        }
    ]
}

```

Create the user interface for the add-in

For the user interface of the add-in, you can use HTML written directly into the **GetDoc_App.html** file. The programming logic and functionality of the add-in must be contained in a JavaScript file (for example, **GetDoc_App.js**).

Use the following procedure to create a simple user interface for the add-in that includes a heading and a single button.

1. In a new file in the text editor, add the HTML for your selected Office application.

PowerPoint

HTML

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=Edge"/>
    <title>Publish presentation</title>
    <link rel="stylesheet" type="text/css" href="Program.css"
  />
    <script src="https://ajax.aspnetcdn.com/ajax/jquery/jquery-
1.9.0.min.js" type="text/javascript"></script>
    <script
src="https://appsforoffice.microsoft.com/lib/1/hosted/office.js"
type="text/javascript"></script>
    <script src="GetDoc_App.js"></script>
  </head>
  <body>
    <form>
      <h1>Publish presentation</h1>
      <br />
      <div><input id='submit' type="button" value="Submit" />
</div>
      <br />
      <div><h2>Status</h2>
        <div id="status"></div>
      </div>
    </form>
  </body>
</html>
```

2. Save the file as **GetDoc_App.html** using UTF-8 encoding to a network location or to a web server.

⚠ **Note**

Be sure that the **head** tags of the add-in contains a **script** tag with a valid link to the Office.js file.

3. We'll use some CSS to give the add-in a simple yet modern and professional appearance. Use the following CSS to define the style of the add-in.

In a new file in the text editor, add the following CSS.

```
css

body
{
    font-family: "Segoe UI Light", "Segoe UI", Tahoma, sans-serif;
}
h1, h2
{
    text-decoration-color: #4ec724;
}
input [type="submit"], input[type="button"]
{
    height: 24px;
    padding-left: 1em;
    padding-right: 1em;
    background-color: white;
    border: 1px solid grey;
    border-color: #dedfe0 #b9b9b9 #b9b9b9 #dedfe0;
    cursor: pointer;
}
```

4. Save the file as **Program.css** using UTF-8 encoding to the network location or to the web server where the **GetDoc_App.html** file is located.

Add the JavaScript to get the document

In the code for the add-in, a handler to the `Office.initialize` event adds a handler to the click event of the **Submit** button on the form and informs the user that the add-in is ready.

The following code example shows the event handler for the `Office.initialize` event along with a helper function, `updateStatus`, for writing to the status div.

```

// The initialize or onReady function is required for all add-ins.
Office.initialize = function (reason) {

    // Checks for the DOM to load using the jQuery ready method.
    $(document).ready(function () {

        // Run sendFile when Submit is clicked.
        $('#submit').on("click", function () {
            sendFile();
        });

        // Update status.
        updateStatus("Ready to send file.");
    });
}

// Create a function for writing to the status div.
function updateStatus(message) {
    var statusInfo = $('#status');
    statusInfo[0].innerHTML += message + "<br/>";
}

```

When you choose the **Submit** button in the UI, the add-in calls the `sendFile` function, which contains a call to the `Document.getFileAsync` method. The `getFileAsync` method uses the asynchronous pattern, similar to other methods in the Office JavaScript API. It has one required parameter, *fileType*, and two optional parameters, *options* and *callback*.

The *fileType* parameter expects one of three constants from the `FileType` enumeration:

`Office.FileType.Compressed` ("compressed"), `Office.FileType.PDF` ("pdf"), or `Office.FileType.Text` ("text"). The current file type support for each platform is listed under the `Document.getFileType` remarks. When you pass in **Compressed** for the *fileType* parameter, the `getFileAsync` method returns the current document as a PowerPoint presentation file (*.pptx) or Word document file (*.docx) by creating a temporary copy of the file on the local computer.

The `getFileAsync` method returns a reference to the file as a `File` object. The `File` object exposes the following four members.

- `size` property
- `sliceCount` property
- `getSliceAsync` method
- `closeAsync` method

The `size` property returns the number of bytes in the file. The `sliceCount` returns the number of `Slice` objects (discussed later in this article) in the file.

Use the following code to get the current PowerPoint or Word document as a `File` object using the `Document.getFileAsync` method and then make a call to the locally defined `getSlice` function. Note that the `File` object, a counter variable, and the total number of slices in the file are passed along in the call to `getSlice` in an anonymous object.

JavaScript

```
// Get all of the content from a PowerPoint or Word document in 100-KB
chunks of text.
function sendFile() {
    Office.context.document.getFileAsync("compressed",
        { sliceSize: 100000 },
        function (result) {

            if (result.status === Office.AsyncResultStatus.Succeeded) {

                // Get the File object from the result.
                var myFile = result.value;
                var state = {
                    file: myFile,
                    counter: 0,
                    sliceCount: myFile.sliceCount
                };

                updateStatus("Getting file of " + myFile.size + " bytes");
                getSlice(state);
            } else {
                updateStatus(result.status);
            }
        });
}
```

The local function `getSlice` makes a call to the `File.getSliceAsync` method to retrieve a slice from the `File` object. The `getSliceAsync` method returns a `Slice` object from the collection of slices. It has two required parameters, *sliceIndex* and *callback*. The *sliceIndex* parameter takes an integer as an indexer into the collection of slices. Like other methods in the Office JavaScript API, the `getSliceAsync` method also takes a callback function as a parameter to handle the results from the method call.

The `Slice` object gives you access to the data contained in the file. Unless otherwise specified in the *options* parameter of the `getFileAsync` method, the `Slice` object is 4 MB in size. The `Slice` object exposes three properties: *size*, *data*, and *index*. The *size* property gets the size, in bytes, of the slice. The *index* property gets an integer that represents the slice's position in the collection of slices.

JavaScript

```
// Get a slice from the file and then call sendSlice.
function getSlice(state) {
    state.file.getSliceAsync(state.counter, function (result) {
        if (result.status == Office.AsyncResultStatus.Succeeded) {
            updateStatus("Sending piece " + (state.counter + 1) + " of " +
state.sliceCount);
            sendSlice(result.value, state);
        } else {
            updateStatus(result.status);
        }
    });
}
```

The `Slice.data` property returns the raw data of the file as a byte array. If the data is in text format (that is, XML or plain text), the slice contains the raw text. If you pass in **Office.FileType.Compressed** for the *fileType* parameter of `Document.getFileAsync`, the slice contains the binary data of the file as a byte array. In the case of a PowerPoint or Word file, the slices contain byte arrays.

You must implement your own function (or use an available library) to convert byte array data to a Base64-encoded string. For information about Base64 encoding with JavaScript, see [Base64 encoding and decoding](#).

Once you've converted the data to Base64, you can then transmit it to a web server in several ways, including as the body of an HTTP POST request.

Add the following code to send a slice to a web service.

ⓘ Note

This code sends a PowerPoint or Word file to the web server in multiple slices. The web server or service must append each individual slice into a single file, and then save it as a .pptx or .docx file before you can perform any manipulations on it.

JavaScript

```
function sendSlice(slice, state) {
    var data = slice.data;

    // If the slice contains data, create an HTTP request.
    if (data) {

        // Encode the slice data, a byte array, as a Base64 string.
        // NOTE: The implementation of myEncodeBase64(input) function isn't
        // included with this example. For information about Base64 encoding
        with
```

```

    // JavaScript, see
    https://developer.mozilla.org/docs/Web/JavaScript/Base64_encoding_and_decodi
    ng.

    var fileData = myEncodeBase64(data);

    // Create a new HTTP request. You need to send the request
    // to a webpage that can receive a post.
    var request = new XMLHttpRequest();

    // Create a handler function to update the status
    // when the request has been sent.
    request.onreadystatechange = function () {
        if (request.readyState == 4) {

            updateStatus("Sent " + slice.size + " bytes.");
            state.counter++;

            if (state.counter < state.sliceCount) {
                getSlice(state);
            } else {
                closeFile(state);
            }
        }
    }

    request.open("POST", "[Your receiving page or service]");
    request.setRequestHeader("Slice-Number", slice.index);

    // Send the file as the body of an HTTP POST
    // request to the web server.
    request.send(fileData);
}
}

```

As the name implies, the `File.closeAsync` method closes the connection to the document and frees up resources. Although the Office Add-ins sandbox garbage collects out-of-scope references to files, it's still a best practice to explicitly close files once your code is done with them. The `closeAsync` method has a single parameter, *callback*, that specifies the function to call on the completion of the call.

JavaScript

```

function closeFile(state) {
    // Close the file when you're done with it.
    state.file.closeAsync(function (result) {

        // If the result returns as a success, the
        // file has been successfully closed.
        if (result.status === Office.AsyncResultStatus.Succeeded) {
            updateStatus("File closed.");
        } else {
            updateStatus("File couldn't be closed.");
        }
    });
}

```

```

    }
  });
}

```

The final JavaScript file could look like the following:

JavaScript

```

/*
 * Copyright (c) Microsoft Corporation. All rights reserved. Licensed under
 the MIT license.
 * See LICENSE in the project root for license information.
 */

// The initialize or onReady function is required for all add-ins.
Office.initialize = function (reason) {

    // Checks for the DOM to load using the jQuery ready method.
    $(document).ready(function () {

        // Run sendFile when Submit is clicked.
        $('#submit').on("click", function () {
            sendFile();
        });

        // Update status.
        updateStatus("Ready to send file.");
    });
}

// Create a function for writing to the status div.
function updateStatus(message) {
    var statusInfo = $('#status');
    statusInfo[0].innerHTML += message + "<br/>";
}

// Get all of the content from a PowerPoint or Word document in 100-KB
chunks of text.
function sendFile() {
    Office.context.document.getFileAsync("compressed",
        { sliceSize: 100000 },
        function (result) {

            if (result.status === Office.AsyncResultStatus.Succeeded) {

                // Get the File object from the result.
                var myFile = result.value;
                var state = {
                    file: myFile,
                    counter: 0,
                    sliceCount: myFile.sliceCount
                };
            }
        }
    );
}

```

```

        updateStatus("Getting file of " + myFile.size + " bytes");
        getSlice(state);
    } else {
        updateStatus(result.status);
    }
});
}

// Get a slice from the file and then call sendSlice.
function getSlice(state) {
    state.file.getSliceAsync(state.counter, function (result) {
        if (result.status == Office.AsyncResultStatus.Succeeded) {
            updateStatus("Sending piece " + (state.counter + 1) + " of " +
state.sliceCount);
            sendSlice(result.value, state);
        } else {
            updateStatus(result.status);
        }
    });
}

function sendSlice(slice, state) {
    var data = slice.data;

    // If the slice contains data, create an HTTP request.
    if (data) {

        // Encode the slice data, a byte array, as a Base64 string.
        // NOTE: The implementation of myEncodeBase64(input) function isn't
        // included with this example. For information about Base64 encoding
with
        // JavaScript, see
https://developer.mozilla.org/docs/Web/JavaScript/Base64\_encoding\_and\_decodi
ng.
        var fileData = myEncodeBase64(data);

        // Create a new HTTP request. You need to send the request
        // to a webpage that can receive a post.
        var request = new XMLHttpRequest();

        // Create a handler function to update the status
        // when the request has been sent.
        request.onreadystatechange = function () {
            if (request.readyState == 4) {

                updateStatus("Sent " + slice.size + " bytes.");
                state.counter++;

                if (state.counter < state.sliceCount) {
                    getSlice(state);
                } else {
                    closeFile(state);
                }
            }
        }
    }
}

```

```
request.open("POST", "[Your receiving page or service]");
request.setRequestHeader("Slice-Number", slice.index);

// Send the file as the body of an HTTP POST
// request to the web server.
request.send(fileData);
}
}

function closeFile(state) {
    // Close the file when you're done with it.
    state.file.closeAsync(function (result) {

        // If the result returns as a success, the
        // file has been successfully closed.
        if (result.status === Office.AsyncResultStatus.Succeeded) {
            updateStatus("File closed.");
        } else {
            updateStatus("File couldn't be closed.");
        }
    });
}
```

Use custom tags for presentations, slides, and shapes in PowerPoint

Article • 07/21/2022

An add-in can attach custom metadata, in the form of key-value pairs, called "tags", to presentations, specific slides, and specific shapes on a slide.

There are two main scenarios for using tags:

- When applied to a slide or a shape, a tag enables the object to be categorized for batch processing. For example, suppose a presentation has some slides that should be included in presentations to the East region but not the West region. Similarly, there are alternative slides that should be shown only to the West. Your add-in can create a tag with the key `REGION` and the value `East` and apply it to the slides that should only be used in the East. The tag's value is set to `West` for the slides that should only be shown to the West region. Just before a presentation to the East, a button in the add-in runs code that loops through all the slides checking the value of the `REGION` tag. Slides where the region is `West` are deleted. The user then closes the add-in and starts the slide show.
- When applied to a presentation, a tag is effectively a custom property in the presentation document (similar to a [CustomProperty](#) in Word).

Tag slides and shapes

A tag is a key-value pair, where the value is always of type `string` and is represented by a [Tag](#) object. Each type of parent object, such as a [Presentation](#), [Slide](#), or [Shape](#) object, has a `tags` property of type [TagsCollection](#).

Add, update, and delete tags

To add a tag to an object, call the [TagCollection.add](#) method of the parent object's `tags` property. The following code adds two tags to the first slide of a presentation. About this code, note:

- The first parameter of the `add` method is the key in the key-value pair.
- The second parameter is the value.
- The key is in uppercase letters. This isn't strictly mandatory for the `add` method; however, the key is always stored by PowerPoint as uppercase, and *some tag-related methods do require that the key be expressed in uppercase*, so we

recommend as a best practice that you always use uppercase in your code for a tag key.

JavaScript

```
async function addMultipleSlideTags() {
    await PowerPoint.run(async function(context) {
        const slide = context.presentation.slides.getItemAt(0);
        slide.tags.add("OCEAN", "Arctic");
        slide.tags.add("PLANET", "Jupiter");

        await context.sync();
    });
}
```

The `add` method is also used to update a tag. The following code changes the value of the `PLANET` tag.

JavaScript

```
async function updateTag() {
    await PowerPoint.run(async function(context) {
        const slide = context.presentation.slides.getItemAt(0);
        slide.tags.add("PLANET", "Mars");

        await context.sync();
    });
}
```

To delete a tag, call the `delete` method on its parent `TagsCollection` object and pass the key of the tag as the parameter. For an example, see [Set custom metadata on the presentation](#).

Use tags to selectively process slides and shapes

Consider the following scenario: Contoso Consulting has a presentation they show to all new customers. But some slides should only be shown to customers that have paid for "premium" status. Before showing the presentation to non-premium customers, they make a copy of it and delete the slides that only premium customers should see. An add-in enables Contoso to tag which slides are for premium customers and to delete these slides when needed. The following list outlines the major coding steps to create this functionality.

1. Create a function that tags the currently selected slide as intended for `Premium` customers. About this code, note:

- The `getSelectedSlideIndex` function is defined in the next step. It returns the 1-based index of the currently selected slide.
- The value returned by the `getSelectedSlideIndex` function has to be decremented because the `SlideCollection.getItemAt` method is 0-based.

JavaScript

```
async function addTagToSelectedSlide() {
    await PowerPoint.run(async function(context) {
        let selectedSlideIndex = await getSelectedSlideIndex();
        selectedSlideIndex = selectedSlideIndex - 1;
        const slide =
context.presentation.slides.getItemAt(selectedSlideIndex);
        slide.tags.add("CUSTOMER_TYPE", "Premium");

        await context.sync();
    });
}
```

2. The following code creates a method to get the index of the selected slide. About this code, note:

- It uses the `Office.context.document.getSelectedDataAsync` method of the Common JavaScript APIs.
- The call to `getSelectedDataAsync` is embedded in a promise-returning function. For more information about why and how to do this, see [Wrap Common APIs in promise-returning functions](#).
- `getSelectedDataAsync` returns an array because multiple slides can be selected. In this scenario, the user has selected just one, so the code gets the first (0th) slide, which is the only one selected.
- The `index` value of the slide is the 1-based value the user sees beside the slide in the PowerPoint UI thumbnails pane.

JavaScript

```
function getSelectedSlideIndex() {
    return new OfficeExtension.Promise<number>(function(resolve,
reject) {

Office.context.document.getSelectedDataAsync(Office.CoercionType.SlideR
ange, function(asyncResult) {
        try {
            if (asyncResult.status ===
Office.AsyncResultStatus.Failed) {
                reject(console.error(asyncResult.error.message));
            } else {
                resolve(asyncResult.value.slides[0].index);
            }
        }
    });
}
```

```

    }
  }
  catch (error) {
    reject(console.log(error));
  }
});
});
}

```

3. The following code creates a function to delete slides that are tagged for premium customers. About this code, note:

- Because the `key` and `value` properties of the tags are going to be read after the `context.sync`, they must be loaded first.

JavaScript

```

async function deleteSlidesByAudience() {
  await PowerPoint.run(async function(context) {
    const slides = context.presentation.slides;
    slides.load("tags/key, tags/value");

    await context.sync();

    for (let i = 0; i < slides.items.length; i++) {
      let currentSlide = slides.items[i];
      for (let j = 0; j < currentSlide.tags.items.length; j++) {
        let currentTag = currentSlide.tags.items[j];
        if (currentTag.key === "CUSTOMER_TYPE" && currentTag.value ===
"Premium") {
          currentSlide.delete();
        }
      }
    }

    await context.sync();
  });
}

```

Set custom metadata on the presentation

Add-ins can also apply tags to the presentation as a whole. This enables you to use tags for document-level metadata similar to how the `CustomProperty` class is used in Word. But unlike the Word `CustomProperty` class, the value of a PowerPoint tag can only be of type `string`.

The following code is an example of adding a tag to a presentation.

JavaScript

```
async function addPresentationTag() {  
  await PowerPoint.run(async function (context) {  
    let presentationTags = context.presentation.tags;  
    presentationTags.add("SECURITY", "Internal-Audience-Only");  
  
    await context.sync();  
  });  
}
```

The following code is an example of deleting a tag from a presentation. Note that the key of the tag is passed to the `delete` method of the parent `TagsCollection` object.

JavaScript

```
async function deletePresentationTag() {  
  await PowerPoint.run(async function (context) {  
    let presentationTags = context.presentation.tags;  
    presentationTags.delete("SECURITY");  
  
    await context.sync();  
  });  
}
```

Use document themes in your PowerPoint add-ins

Article • 06/18/2024

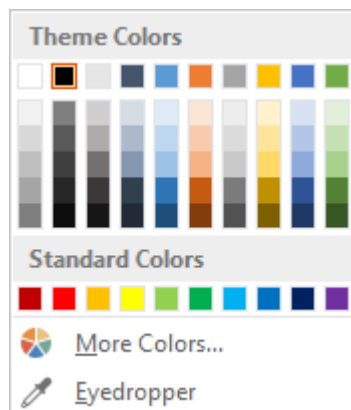
An [Office theme](#) consists, in part, of a visually coordinated set of fonts and colors that you can apply to presentations, documents, worksheets, and emails. To apply or customize the theme of a presentation in PowerPoint, you use the **Themes** and **Variants** groups on **Design** tab of the ribbon. PowerPoint assigns a new blank presentation with the default **Office Theme**, but you can choose other themes available on the **Design** tab, download additional themes from Office.com, or create and customize your own theme.

Using **OfficeThemes.css**, design add-ins that are coordinated with PowerPoint in two ways.

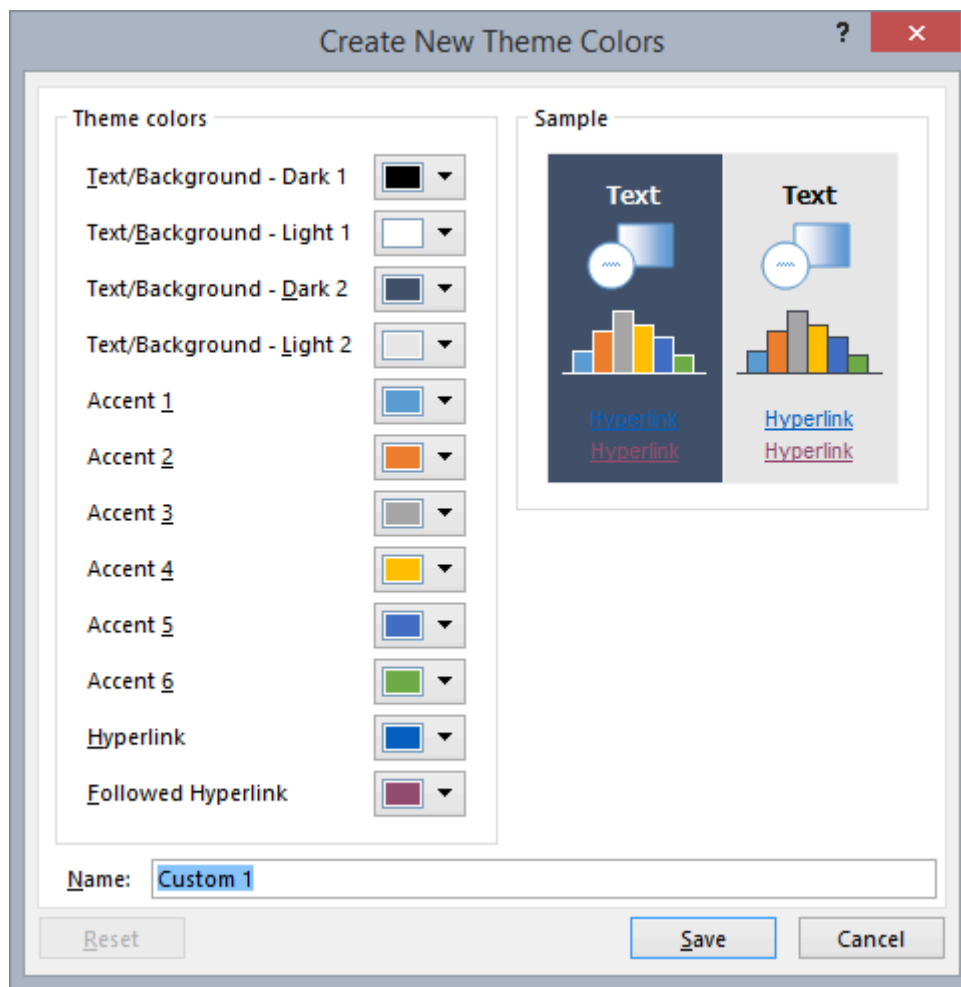
- **In content add-ins for PowerPoint.** Use the document theme classes of **OfficeThemes.css** to specify fonts and colors that match the theme of the presentation your content add-in is inserted into - and those fonts and colors will dynamically update if a user changes or customizes the presentation's theme.
- **In task pane add-ins for PowerPoint.** Use the Office UI theme classes of **OfficeThemes.css** to specify the same fonts and background colors used in the UI so that your task pane add-ins will match the colors of built-in task panes - and those colors will dynamically update if a user changes the Office UI theme.

Document theme colors

Every Office document theme defines 12 colors. Ten of these colors are available when you set font, background, and other color settings in a presentation with the color picker.



To view or customize the full set of 12 theme colors in PowerPoint, in the **Variants** group on the **Design** tab, click the **More** drop-down - then select **Colors > Customize Colors** to display the **Create New Theme Colors** dialog box.



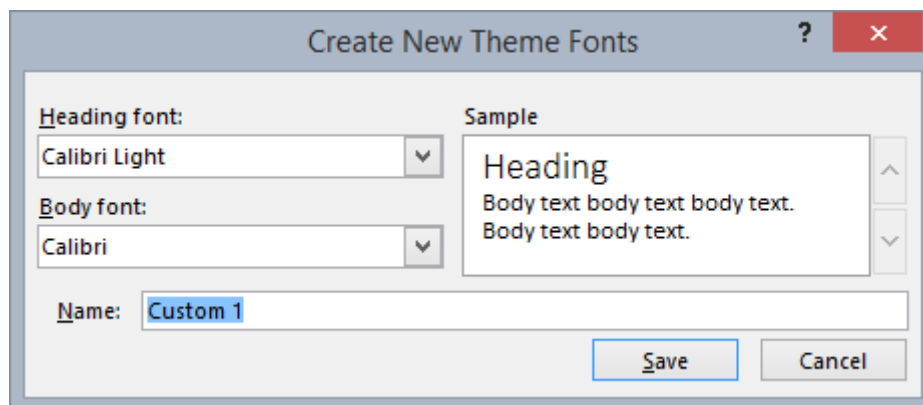
The first four colors are for text and backgrounds. Text that is created with the light colors will always be legible over the dark colors, and text that is created with dark colors will always be legible over the light colors. The next six are accent colors that are always visible over the four potential background colors. The last two colors are for hyperlinks and followed hyperlinks.

Document theme fonts

Every Office document theme also defines two fonts -- one for headings and one for body text. PowerPoint uses these fonts to construct automatic text styles. In addition, **Quick Styles** galleries for text and **WordArt** use these same theme fonts. These two fonts are available as the first two selections when you select fonts with the font picker.



To view or customize theme fonts in PowerPoint, in the **Variants** group on the **Design** tab, click the **More** drop-down - then select **Fonts > Customize Fonts** to display the **Create New Theme Fonts** dialog box.



Office UI theme fonts and colors

Office also lets you choose between several predefined themes that specify some of the colors and fonts used in the UI of all Office applications. To do that, you use the **File > Account > Office Theme** drop-down (from any Office application).

Office Theme:



OfficeThemes.css includes classes that you can use in your task pane add-ins for PowerPoint so they will use these same fonts and colors. This lets you design your task pane add-ins that match the appearance of built-in task panes.

Use OfficeThemes.css

Using the **OfficeThemes.css** file with your content add-ins for PowerPoint lets you coordinate the appearance of your add-in with the theme applied to the presentation

it's running with. Using the **OfficeThemes.css** file with your task pane add-ins for PowerPoint lets you coordinate the appearance of your add-in with the fonts and colors of the Office UI.

Add the OfficeThemes.css file to your project

Use the following steps to add and reference the **OfficeThemes.css** file to your add-in project.

ⓘ Note

The steps in this procedure only apply to Visual Studio 2015. If you are using Visual Studio 2019, the **OfficeThemes.css** file is created automatically for any new PowerPoint add-in projects that you create.

1. In **Solution Explorer**, right-click (or select and hold) the **Content** folder in the *project_nameWeb* project, choose **Add**, and then select **Style Sheet**.
2. Name the new style sheet **OfficeThemes**.

ⓘ Important

The style sheet must be named **OfficeThemes**, or the feature that dynamically updates add-in fonts and colors when a user changes the theme won't work.

3. Delete the default **body** class (`body {}`) in the file, and copy and paste the following CSS code into the file.

CSS

```
/* The following classes describe the common theme information for
office documents */

/* Basic Font and Background Colors for text */
.office-docTheme-primary-fontColor { color:#000000; }
.office-docTheme-primary-bgColor { background-color:#ffffff; }
.office-docTheme-secondary-fontColor { color: #000000; }
.office-docTheme-secondary-bgColor { background-color: #ffffff; }

/* Accent color definitions for fonts */
.office-contentAccent1-color { color:#5b9bd5; }
.office-contentAccent2-color { color:#ed7d31; }
.office-contentAccent3-color { color:#a5a5a5; }
.office-contentAccent4-color { color:#ffc000; }
```

```

.office-contentAccent5-color { color:#4472c4; }
.office-contentAccent6-color { color:#70ad47; }

/* Accent color for backgrounds */
.office-contentAccent1-bgColor { background-color:#5b9bd5; }
.office-contentAccent2-bgColor { background-color:#ed7d31; }
.office-contentAccent3-bgColor { background-color:#a5a5a5; }
.office-contentAccent4-bgColor { background-color:#ffc000; }
.office-contentAccent5-bgColor { background-color:#4472c4; }
.office-contentAccent6-bgColor { background-color:#70ad47; }

/* Accent color for borders */
.office-contentAccent1-borderColor { border-color:#5b9bd5; }
.office-contentAccent2-borderColor { border-color:#ed7d31; }
.office-contentAccent3-borderColor { border-color:#a5a5a5; }
.office-contentAccent4-borderColor { border-color:#ffc000; }
.office-contentAccent5-borderColor { border-color:#4472c4; }
.office-contentAccent6-borderColor { border-color:#70ad47; }

/* links */
.office-a { color: #0563c1; }
.office-a:visited { color: #954f72; }

/* Body Fonts */
.office-bodyFont-eastAsian { } /* East Asian name of the Font */
.office-bodyFont-latin { font-family:"Calibri"; } /* Latin name of the
Font */
.office-bodyFont-script { } /* Script name of the Font */
.office-bodyFont-localized { font-family:"Calibri"; } /* Localized name
of the Font. Corresponds to the default font of the culture currently
used in Office.*/

/* Headers Font */
.office-headerFont-eastAsian { }
.office-headerFont-latin { font-family:"Calibri Light"; }
.office-headerFont-script { }
.office-headerFont-localized { font-family:"Calibri Light"; }

/* The following classes define font and background colors for Office
UI themes. These classes should only be used in task pane add-ins */

/* Basic Font and Background Colors for PPT */
.office-officeTheme-primary-fontColor { color:#b83b1d; }
.office-officeTheme-primary-bgColor { background-color:#dedede; }
.office-officeTheme-secondary-fontColor { color:#262626; }
.office-officeTheme-secondary-bgColor { background-color:#ffffff; }

```

4. If you are using a tool other than Visual Studio to create your add-in, copy the CSS code from the previous step into a text file. Then, save the file as **OfficeThemes.css**.

Reference OfficeThemes.css in your add-in's HTML pages

To use the **OfficeThemes.css** file in your add-in project, add a `<link>` tag that references the **OfficeThemes.css** file inside the `<head>` tag of the web pages (such as an .html, .aspx, or .php file) that implement the UI of your add-in in this format.

HTML

```
<link href="<local_path_to_OfficeThemes.css>" rel="stylesheet" type="text/css" />
```

To do this in Visual Studio, follow these steps.

1. 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.
6. In the HTML pages that implement the UI of your add-in, such as Home.html in the default template, add the following `<link>` tag inside the `<head>` tag that references the **OfficeThemes.css** file.

HTML

```
<link href="../../../Content/OfficeThemes.css" rel="stylesheet" type="text/css" />
```

If you are creating your add-in with a tool other than Visual Studio, add a `<link>` tag with the same format specifying a relative path to the copy of **OfficeThemes.css** that will be deployed with your add-in.

Use OfficeThemes.css document theme classes in your content add-in's HTML page

The following shows a simple example of HTML in a content add-in that uses the OfficeTheme.css document theme classes. For details about the **OfficeThemes.css** classes that correspond to the 12 colors and 2 fonts used in a document theme, see [Theme classes for content add-ins](#).

HTML

```
<body>
  <div id="themeSample" class="office-docTheme-primary-fontColor ">
    <h1 class="office-headerFont-latin">Hello world!</h1>
    <h1 class="office-headerFont-latin office-contentAccent1-
bgColor">Hello world!</h1>
    <h1 class="office-headerFont-latin office-contentAccent2-
bgColor">Hello world!</h1>
    <h1 class="office-headerFont-latin office-contentAccent3-
bgColor">Hello world!</h1>
    <h1 class="office-headerFont-latin office-contentAccent4-
bgColor">Hello world!</h1>
    <h1 class="office-headerFont-latin office-contentAccent5-
bgColor">Hello world!</h1>
    <h1 class="office-headerFont-latin office-contentAccent6-
bgColor">Hello world!</h1>
    <p class="office-bodyFont-latin office-docTheme-secondary-
fontColor">Hello world!</p>
  </div>
</body>
```

At runtime, when inserted into a presentation that uses the default **Office Theme**, the content add-in is rendered like this.

Hello world!

Hello world!

Hello world!

Hello world!

Hello world!

Hello world!

Hello world!

Hello world!

If you change the presentation to use another theme or customize the presentation's theme, the fonts and colors specified with **OfficeThemes.css** classes will dynamically update to correspond to the fonts and colors of the presentation's theme. Using the

same HTML example as above, if the presentation the add-in is inserted into uses the **Facet** theme, the add-in rendering will look like this.

Hello world!

Hello world!

Hello world!

Hello world!

Hello world!

Hello world!

Hello world!

Hello world!

Use OfficeThemes.css Office UI theme classes in your task pane add-in's HTML page

In addition to the document theme, users can customize the color scheme of the Office user interface for all Office applications using the **File > Account > Office Theme** drop-down box.

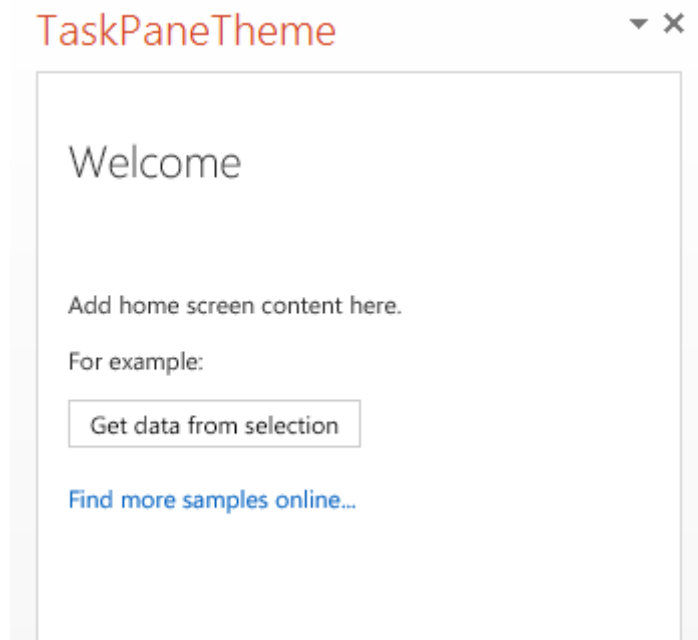
The following shows a simple example of HTML in a task pane add-in that uses OfficeTheme.css classes to specify font color and background color. For details about the **OfficeThemes.css** classes that correspond to fonts and colors of the Office UI theme, see [Theme classes for task pane add-ins](#).

HTML

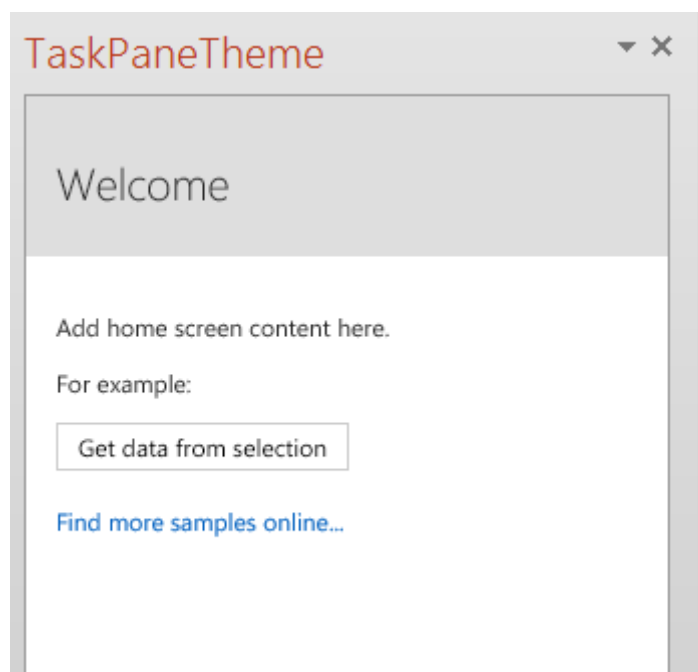
```
<body>
  <div id="content-header" class="office-officeTheme-primary-fontColor
office-officeTheme-primary-bgColor">
    <div class="padding">
      <h1>Welcome</h1>
    </div>
  </div>
  <div id="content-main" class="office-officeTheme-secondary-fontColor
office-officeTheme-secondary-bgColor">
    <div class="padding">
      <p>Add home screen content here.</p>
      <p>For example:</p>
      <button id="get-data-from-selection">Get data from
selection</button>
      <p><a target="_blank" class="office-a"
```

```
href="https://go.microsoft.com/fwlink/?LinkId=276812">Find more samples  
online...</a></p>  
    </div>  
</div>  
</body>
```

When running in PowerPoint with **File > Account > Office Theme** set to **White**, the task pane add-in is rendered like this.



If you change **OfficeTheme** to **Dark Gray**, the fonts and colors specified with **OfficeThemes.css** classes will dynamically update to render like this.



OfficeTheme.css classes

The **OfficeThemes.css** file contains two sets of classes you can use with your content and task pane add-ins for PowerPoint.

Theme classes for content add-ins

The **OfficeThemes.css** file provides classes that correspond to the 2 fonts and 12 colors used in a document theme. These classes are appropriate to use with content add-ins for PowerPoint so that your add-in's fonts and colors will be coordinated with the presentation it's inserted into.

Theme fonts for content add-ins

 Expand table

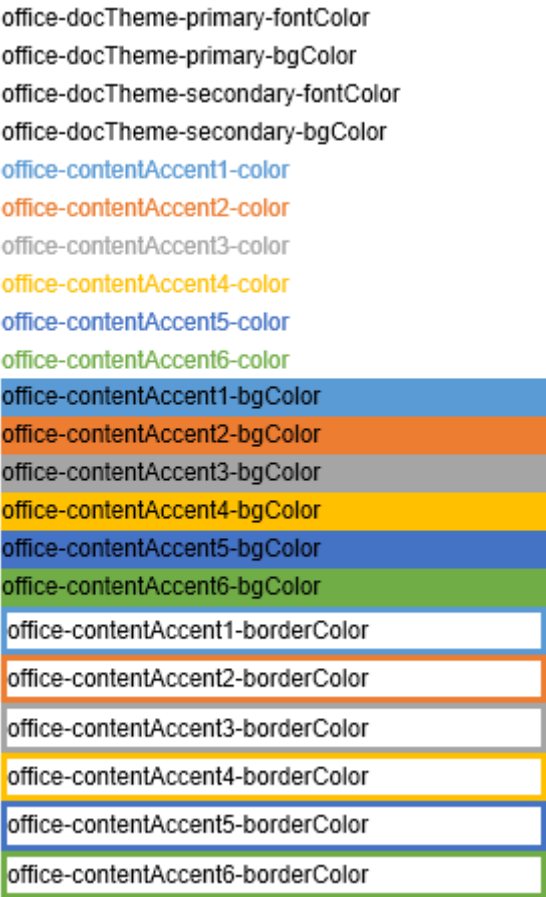
Class	Description
<code>office-bodyFont-eastAsian</code>	East Asian name of the body font.
<code>office-bodyFont-latin</code>	Latin name of the body font. Default "Calibri"
<code>office-bodyFont-script</code>	Script name of the body font.
<code>office-bodyFont-localized</code>	Localized name of the body font. Specifies the default font name according to the culture currently used in Office.
<code>office-headerFont-eastAsian</code>	East Asian name of the headers font.
<code>office-headerFont-latin</code>	Latin name of the headers font. Default "Calibri Light"
<code>office-headerFont-script</code>	Script name of the headers font.
<code>office-headerFont-localized</code>	Localized name of the headers font. Specifies the default font name according to the culture currently used in Office.

Theme colors for content add-ins

 Expand table

Class	Description
office-docTheme-primary-fontColor	Primary font color. Default #000000
office-docTheme-primary-bgColor	Primary font background color. Default #FFFFFF
office-docTheme-secondary-fontColor	Secondary font color. Default #000000
office-docTheme-secondary-bgColor	Secondary font background color. Default #FFFFFF
office-contentAccent1-color	Font accent color 1. Default #5B9BD5
office-contentAccent2-color	Font accent color 2. Default #ED7D31
office-contentAccent3-color	Font accent color 3. Default #A5A5A5
office-contentAccent4-color	Font accent color 4. Default #FFC000
office-contentAccent5-color	Font accent color 5. Default #4472C4
office-contentAccent6-color	Font accent color 6. Default #70AD47
office-contentAccent1-bgColor	Background accent color 1. Default #5B9BD5
office-contentAccent2-bgColor	Background accent color 2. Default #ED7D31
office-contentAccent3-bgColor	Background accent color 3. Default #A5A5A5
office-contentAccent4-bgColor	Background accent color 4. Default #FFC000
office-contentAccent5-bgColor	Background accent color 5. Default #4472C4
office-contentAccent6-bgColor	Background accent color 6. Default #70AD47
office-contentAccent1-borderColor	Border accent color 1. Default #5B9BD5
office-contentAccent2-borderColor	Border accent color 2. Default #ED7D31
office-contentAccent3-borderColor	Border accent color 3. Default #A5A5A5
office-contentAccent4-borderColor	Border accent color 4. Default #FFC000
office-contentAccent5-borderColor	Border accent color 5. Default #4472C4
office-contentAccent6-borderColor	Border accent color 6. Default #70AD47
office-a	Hyperlink color. Default #0563C1
office-a:visited	Followed hyperlink color. Default #954F72

The following screenshot shows examples of all of the theme color classes (except for the two hyperlink colors) assigned to add-in text when using the default Office theme.



Theme classes for task pane add-ins

The `OfficeThemes.css` file provides classes that correspond to the four colors assigned to fonts and backgrounds used by the Office application UI theme. These classes are appropriate to use with task add-ins for PowerPoint, so that your add-in's colors are coordinated with the other built-in task panes in Office.

Theme font and background colors for task pane add-ins

[Expand table](#)

Class	Description
<code>office-officeTheme-primary-fontColor</code>	Primary font color. Default #B83B1D
<code>office-officeTheme-primary-bgColor</code>	Primary background color. Default #DEDEDE
<code>office-officeTheme-secondary-fontColor</code>	Secondary font color. Default #262626
<code>office-officeTheme-secondary-bgColor</code>	Secondary background color. Default #FFFFFF

See also

- [Create content and task pane add-ins for PowerPoint](#)

Collaborate with us on GitHub

The source for this content can be found on GitHub, where you can also create and review issues and pull requests. For more information, see [our contributor guide](#).

Office Add-ins feedback

Office Add-ins is an open source project. Select a link to provide feedback:

 [Open a documentation issue](#)

 [Provide product feedback](#)

Work with shapes using the PowerPoint JavaScript API

Article • 05/07/2025

This article describes how to use geometric shapes, lines, and text boxes in conjunction with the [Shape](#) and [ShapeCollection](#) APIs.

Create shapes

Shapes are created through and stored in a slide's shape collection (`slide.shapes`).

`ShapeCollection` has several `.add*` methods for this purpose. All shapes have names and IDs generated for them when they are added to the collection. These are the `name` and `id` properties, respectively. `name` can be set by your add-in.

Geometric shapes

A geometric shape is created with one of the overloads of `ShapeCollection.addGeometricShape`. The first parameter is either a [GeometricShapeType](#) enum or the string equivalent of one of the enum's values. There is an optional second parameter of type [ShapeAddOptions](#) that can specify the initial size of the shape and its position relative to the top and left sides of the slide, measured in points. Or these properties can be set after the shape is created.

The following code sample creates a rectangle named **"Square"** that is positioned 100 points from the top and left sides of the slide. The method returns a `Shape` object.

JavaScript

```
// This sample creates a rectangle positioned 100 points from the top and left
// sides
// of the slide and is 150x150 points. The shape is put on the first slide.
await PowerPoint.run(async (context) => {
    const shapes = context.presentation.slides.getItemAt(0).shapes;
    const rectangle =
shapes.addGeometricShape(PowerPoint.GeometricShapeType.rectangle);
    rectangle.left = 100;
    rectangle.top = 100;
    rectangle.height = 150;
    rectangle.width = 150;
    rectangle.name = "Square";
    await context.sync();
});
```

Lines

A line is created with one of the overloads of `ShapeCollection.addLine`. The first parameter is either a `ConnectorType` enum or the string equivalent of one of the enum's values to specify how the line contorts between endpoints. There is an optional second parameter of type `ShapeAddOptions` that can specify the start and end points of the line. Or these properties can be set after the shape is created. The method returns a `Shape` object.

⚠ Note

When the shape is a line, the `top` and `left` properties of the `Shape` and `ShapeAddOptions` objects specify the starting point of the line relative to the top and left edges of the slide. The `height` and `width` properties specify the endpoint of the line *relative to the start point*. So, the end point relative to the top and left edges of the slide is $(\text{top} + \text{height})$ by $(\text{left} + \text{width})$. The unit of measure for all properties is points and negative values are allowed.

The following code sample creates a straight line on the slide.

JavaScript

```
// This sample creates a straight line on the first slide.
await PowerPoint.run(async (context) => {
    const shapes = context.presentation.slides.getItemAt(0).shapes;
    const line = shapes.addLine(PowerPoint.ConnectorType.straight, {left: 200,
top: 50, height: 300, width: 150});
    line.name = "StraightLine";
    await context.sync();
});
```

Text boxes

A text box is created with the `addTextBox` method. The first parameter is the text that should appear in the box initially. There is an optional second parameter of type `ShapeAddOptions` that can specify the initial size of the text box and its position relative to the top and left sides of the slide. Or these properties can be set after the shape is created.

The following code sample shows how to create a text box on the first slide.

JavaScript

```
// This sample creates a text box with the text "Hello!" and sizes it
appropriately.
```

```
await PowerPoint.run(async (context) => {
    const shapes = context.presentation.slides.getItemAt(0).shapes;
    const textbox = shapes.addTextBox("Hello!");
    textbox.left = 100;
    textbox.top = 100;
    textbox.height = 300;
    textbox.width = 450;
    textbox.name = "Textbox";
    await context.sync();
});
```

Move and resize shapes

Shapes sit on top of the slide. Their placement is defined by the `left` and `top` properties. These act as margins from slide's respective edges, measured in points, with `left: 0` and `top: 0` being the upper-left corner. The shape size is specified by the `height` and `width` properties. Your code can move or resize the shape by resetting these properties. (These properties have a slightly different meaning when the shape is a line. See [Lines](#).)

Text in shapes

Geometric shapes can contain text. Shapes have a `textFrame` property of type `TextFrame`. The `TextFrame` object manages the text display options (such as margins and text overflow). `TextFrame.textRange` is a `TextRange` object with the text content and font settings.

The following code sample creates a geometric shape named "Braces" with the text "Shape text". It also adjusts the shape and text colors, as well as sets the text's vertical alignment to the center.

JavaScript

```
// This sample creates a light blue rectangle with braces ("{}") on the left and
// right ends
// and adds the purple text "Shape text" to the center.
await PowerPoint.run(async (context) => {
    const shapes = context.presentation.slides.getItemAt(0).shapes;
    const braces =
shapes.addGeometricShape(PowerPoint.GeometricShapeType.bracePair);
    braces.left = 100;
    braces.top = 400;
    braces.height = 50;
    braces.width = 150;
    braces.name = "Braces";
    braces.fill.setSolidColor("lightblue");
    braces.textFrame.textRange.text = "Shape text";
    braces.textFrame.textRange.font.color = "purple";
```

```
braces.textFrame.verticalAlignment =  
PowerPoint.TextVerticalAlignment.middleCentered;  
    await context.sync();  
});
```

Group and ungroup shapes

In PowerPoint, you can group several shapes and treat them like a single shape. You can subsequently ungroup grouped shapes. To learn more about grouping objects in the PowerPoint UI, see [Group or ungroup shapes, pictures, or other objects](#).

Group shapes

To group shapes with the JavaScript API, use [ShapeCollection.addGroup](#).

The following code sample shows how to group existing shapes of type [GeometricShape](#) found on the current slide.

TypeScript

```
// Groups the geometric shapes on the current slide.  
await PowerPoint.run(async (context) => {  
    // Get the shapes on the current slide.  
    context.presentation.load("slides");  
    const slide = context.presentation.getSelectedSlides().getItemAt(0);  
    slide.load("shapes/items/type,shapes/items/id");  
    await context.sync();  
  
    const shapes = slide.shapes;  
    const shapesToGroup = shapes.items.filter((item) => item.type ===  
PowerPoint.ShapeType.geometricShape);  
    if (shapesToGroup.length === 0) {  
        console.warn("No shapes on the current slide, so nothing to group.");  
        return;  
    }  
  
    // Group the geometric shapes.  
    console.log(`Number of shapes to group: ${shapesToGroup.length}`);  
    const group = shapes.addGroup(shapesToGroup);  
    group.load("id");  
    await context.sync();  
  
    console.log(`Grouped shapes. Group ID: ${group.id}`);  
});
```

Ungroup shapes

To ungroup shapes with the JavaScript API, get the `group` property from the group's `Shape` object then call `ShapeGroup.ungroup`.

The following code sample shows how to ungroup the first shape group found on the current slide.

JavaScript

```
// Ungroups the first shape group on the current slide.
await PowerPoint.run(async (context) => {
  // Get the shapes on the current slide.
  context.presentation.load("slides");
  const slide = context.presentation.getSelectedSlides().getItemAt(0);
  slide.load("shapes/items/type,shapes/items/id");
  await context.sync();

  const shapes = slide.shapes;
  const shapeGroups = shapes.items.filter((item) => item.type ===
PowerPoint.ShapeType.group);
  if (shapeGroups.length === 0) {
    console.warn("No shape groups on the current slide, so nothing to
ungroup.");
    return;
  }

  // Ungroup the first grouped shapes.
  const firstGroupId = shapeGroups[0].id;
  const shapeGroupToUngroup = shapes.getItem(firstGroupId);
  shapeGroupToUngroup.group.ungroup();
  await context.sync();

  console.log(`Ungrouped shapes with group ID: ${firstGroupId}`);
});
```

Delete shapes

Shapes are removed from the slide with the `Shape` object's `delete` method.

The following code sample shows how to delete shapes.

JavaScript

```
await PowerPoint.run(async (context) => {
  // Delete all shapes from the first slide.
  const shapes = context.presentation.slides.getItemAt(0).shapes;

  // Load all the shapes in the collection without loading their properties.
  shapes.load("items/$none");
  await context.sync();
});
```

```
shapes.items.forEach(function (shape) {  
    shape.delete();  
});  
await context.sync();  
});
```

See also

- [Work with tables using the PowerPoint JavaScript API](#)
- [Bind to shapes in a PowerPoint presentation](#)
- [Group or ungroup shapes, pictures, or other objects](#) [↗](#)

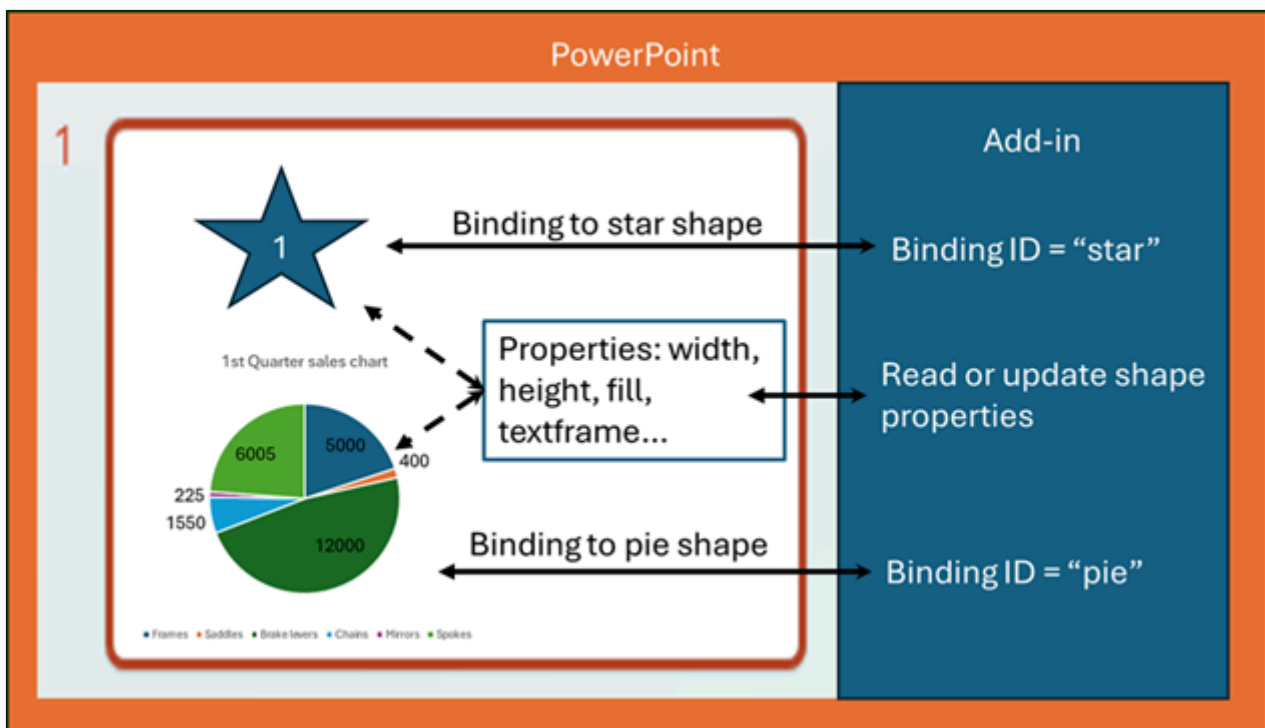
Bind to shapes in a PowerPoint presentation

Article • 04/30/2025

Your PowerPoint add-in can bind to shapes to consistently access them through an identifier. The add-in establishes a binding by calling [BindingCollection.add](#) and assigning a unique identifier. Use the identifier at any time to reference the shape and access its properties. Creating bindings provides the following value to your add-in.

- Establishes a relationship between the add-in and the shape in the document. Bindings are persisted in the document and can be accessed at a later time.
- Enables access to shape properties to read or update, without requiring the user to select any shapes.

The following image shows how an add-in might bind to two shapes on a slide. Each shape has a binding ID created by the add-in: `star` and `pie`. Using the binding ID, the add-in can access the desired shape to update properties.



Scenario: Use bindings to sync with a data source

A common scenario for using bindings is to keep shapes up to date with a data source. Often when creating a presentation, users copy and paste images from the data source into the presentation. Over time, to keep the images up to date, they will manually copy and paste the latest images from the data source. An add-in can help automate this process by retrieving up-to-date images from the data source on the user's behalf. When a shape fill needs updating,

the add-in uses the binding to find the correct shape and update the shape fill with the newer image.

In a general implementation, there are two components to consider for binding a shape in PowerPoint and updating it with a new image from a data source.

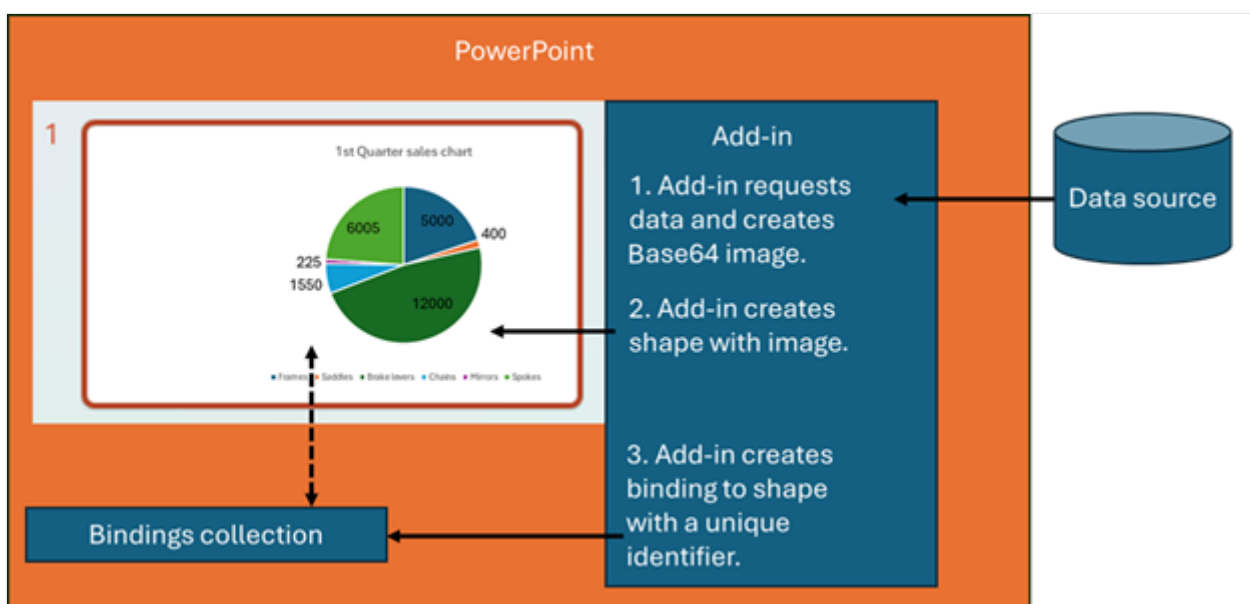
1. **The data source.** This is any source of data or asset library such as Microsoft SharePoint or Microsoft OneDrive.
2. **The PowerPoint add-in.** The add-in gets data from the data source based on what the user needs. It converts the data to a Base64-encoded image. This is the only fill type the bound shape can accept. It inserts a shape upon the user's request and binds it with a unique identifier. Then it fills the shape with the Base64 image based on the original data source. Shapes are updated upon the user's request and the add-in uses the binding identifier to find the shape and update the image with the last saved Base64 image.

ⓘ Note

You decide the implementation details of how to sync updates from the data source and how to get or create images. This article only describes how to use the Office JS APIs in your add-in to bind a shape and update it with latest images.

Create a bound shape in PowerPoint

Use the `PowerPoint.BindingCollection.add()` method for the presentation to create a binding which refers to a particular shape.



The following sample shows how to create a shape on the first selected slide.

JavaScript

```
await PowerPoint.run(async (context) => {
    const slides = context.presentation.getSelectedSlides();

    // Insert new shape on first selected slide.
    const myShape = slides
        .getItemAt(0)
        .shapes.addGeometricShape(PowerPoint.GeometricShapeType.rectangle, {
            top: 100,
            left: 30,
            width: 200,
            height: 200
        });

    // Fill shape with a Base64-encoded image.
    // Note: The image is typically created from a data source request.
    const productsImage = "...base64 image data...";
    myShape.fill.setImage(productsImage);
});
```

Call `BindingCollection.add` to add the binding to the bindings collection in PowerPoint. The following sample shows how to add a new binding for a shape to the bindings collection.

JavaScript

```
// Create a binding ID to track the shape for later updates.
const bindingId = "productChart";
// Create binding by adding the new shape to the bindings collection.
context.presentation.bindings.add(myShape, PowerPoint.BindingType.shape,
bindingId);
```

Refresh a bound shape with updated data

After there's an update to the image data, refresh the shape image by finding it via the binding identifier. The following code sample shows how to find a bound shape with the identifier and fill it with an updated image. The image is updated by the add-in based on the data source request or provided by the data source directly.

JavaScript

```
async function updateBinding(bindingId, image) {
    await PowerPoint.run(async (context) => {
        try {
            // Get the shape based on binding ID.
            const myShape = context.presentation.bindings
                .getItem(bindingId)
                .getShape();
```

```

        // Update the shape to latest image.
        myShape.fill.setImage(image);
        await context.sync();

    } catch (err) {
        console.error(err);
    }
});
}

```

Delete a binding

The following sample shows how to delete a binding by deleting it from the bindings collection.

JavaScript

```

async function deleteBinding(bindingId) {
    await PowerPoint.run(async (context) => {
        context.presentation.bindings.getItemAt(bindingId).delete();
        await context.sync();
    });
}

```

Load bindings

When a user opens a presentation and your add-in first loads, you can load all the bindings to continue working with them. The following code shows how to load all bindings in a presentation and display them in the console.

JavaScript

```

async function loadBindings() {
    await PowerPoint.run(async (context) => {
        try {
            let myBindings = context.presentation.bindings;
            myBindings.load("items");
            await context.sync();

            // Log all binding IDs to console.
            if (myBindings.items.length > 0) {
                myBindings.items.forEach(async (binding) => {
                    console.log(binding.id);
                });
            }
        } catch (err) {

```

```
        console.error(err);
    }
});
}
```

Error handling when a binding or shape is deleted

When a shape is deleted, its associated binding is also removed from the PowerPoint binding collection. Any object references you have to the binding, or shape, will return errors if you access any properties or methods on those objects. Be sure to handle potential error scenarios for a deleted shape if your add-in keeps Binding or Shape objects.

The following code shows one approach to error handling when a binding object references a deleted binding. Use a try/catch statement and then call a function to reload all binding and shape references when an error occurs.

JavaScript

```
async function getShapeFromBindingID(id) {
    await PowerPoint.run(async (context) => {
        try {
            const binding = context.presentation.bindings.getItemAt(id);
            const shape = binding.getShape();

            await context.sync();
            return shape;
        } catch (err) {
            console.log(err);
            return undefined;
        }
    });
}
```

See also

When maintaining freshness on shapes, you may also want to check the `zOrder`. See the [zOrderPosition](#) property for more information.

- [Work with shapes using the PowerPoint JavaScript API](#)
- [Bind to regions in a document or spreadsheet](#)

Work with tables using the PowerPoint JavaScript API

Article • 04/30/2025

This article provides code samples that show how to create tables and control formatting by using the PowerPoint JavaScript API.

Create an empty table

To create an empty table, call the [ShapeCollection.addTable\(\)](#) method and specify how many rows and columns the table needs. The following code sample shows how to create a table with 3 rows and 4 columns.

JavaScript

```
await PowerPoint.run(async (context) => {
    const shapes = context.presentation.getSelectedSlides().getItemAt(0).shapes;

    // Add a table (which is a type of Shape).
    const shape = shapes.addTable(3, 4);
    await context.sync();
});
```

The previous sample doesn't specify any options, so the table defaults to formatting provided by PowerPoint. The following image shows an example of an empty table created with default formatting in PowerPoint.

Specify values

You can populate the table with string values when you create it. To do this provide a 2-dimensional array of values in the [TableAddOptions](#) object. The following code sample creates a table with string values from "1" to "12". Note the following:

- An empty cell must be specified as an empty string "". If a value is undefined or missing, `addTable` throws an error.
- The outer array contains a list of rows. Each row is an inner array containing a list of string cell values.
- The function named `insertTableOnCurrentSlide` is used in other samples in this article.

```

async function run() {
  const options: PowerPoint.TableAddOptions = {
    values: [
      ["1", "2", "", "4"], // Cell 3 is blank.
      ["5", "6", "7", "8"],
      ["9", "10", "11", "12"]
    ],
  };

  await insertTableOnCurrentSlide(3, 4, options);
}

async function insertTableOnCurrentSlide(rowCount: number, columnCount: number,
options: PowerPoint.TableAddOptions) {
  await PowerPoint.run(async (context) => {
    const shapes =
context.presentation.getSelectedSlides().getItemAt(0).shapes;

    // Add a table (which is a type of Shape).
    const shape = shapes.addTable(rowCount, columnCount, options);
    await context.sync();
  });
}

```

The previous sample creates a table with values as shown in the following image.

1	2		4
5	6	7	8
9	10	11	12

Specify cell formatting

You can specify cell formatting when you create a table, including border style, fill style, font style, horizontal alignment, indent level, and vertical alignment. These formats are specified by the [TableCellProperties](#) object.

Uniform cell formatting

Uniform cell formatting applies to the entire table. For example, if you set the uniform font color to white, all table cells will use the white font. Uniform cell formatting is useful for controlling the default formatting you want on the entire table.

Specify uniform cell formatting for the entire table using the [TableAddOptions.uniformCellProperties](#) property. The following code sample shows how to set all table cells to dark slate blue fill color and bold white font.

JavaScript

```
const rowCount = 3;
const columnCount = 4;
const options: PowerPoint.TableAddOptions = {
  values: [
    ["1", "2", "", "4"],
    ["5", "6", "7", "8"],
    ["9", "10", "11", "12"]
  ],
  uniformCellProperties: {
    fill: { color: "darkslateblue" },
    font: { bold: true, color: "white" }
  }
};
await insertTableOnCurrentSlide(rowCount, columnCount, options);
```

The previous sample creates a table as shown in the following image.

1	2		4
5	6	7	8
9	10	11	12

Specific cell formatting

Specific cell formatting applies to individual cells and overrides the uniform cell formatting, if any. Set individual cell formatting by using the [TableAddOptions.specificCellProperties](#) property. The following code sample shows how to set the fill color to black for the cell at row 1, column 1.

Note the `specificCellProperties` must be a 2D array that matches the 2D size of the table exactly. The sample first creates the entire empty 2D array of objects. Then it sets the specific cell format at row 1, column 1, after the options object is created.

JavaScript

```
const rowCount = 3;
const columnCount = 4;
// Compact syntax to create a 2D array filled with empty and distinct objects.
const specificCellProperties = Array(rowCount).fill("").map(_ =>
Array(columnCount).fill("").map(_ => ({})));
const options: PowerPoint.TableAddOptions = {
  values: [
    ["1", "2", "", "4"],
    ["5", "6", "7", "8"],
    ["9", "10", "11", "12"]
  ],
```

```

uniformCellProperties: {
  fill: { color: "darkslateblue" },
  font: { bold: true, color: "white" }
},
specificCellProperties // Array values are empty objects at this point.
};
// Set fill color for specific cell at row 1, column 1.
options.specificCellProperties[1][1] = {
  fill: { color: "black" }
};
await insertTableOnCurrentSlide(rowCount, columnCount, options);

```

The previous sample creates a table with a specific format applied to the cell in row 1, column 1 as shown in the following image.

1	2	4
5	6	7
9	10	11
		12

The previous sample uses the `font` property which is of type `FontProperties`. The `font` property allows you to specify many properties, such as bold, italic, name, color, and more. The following code sample shows how to specify multiple properties for a font for a cell.

JavaScript

```

options.specificCellProperties[1][1] = {
  font: {
    color: "orange",
    name: "Arial",
    size: 50,
    allCaps: true,
    italic: true
  }
};

```

You can also specify a `fill` property which is of type `FillProperties`. The `fill` property can specify a color and the transparency percentage. The following code sample shows how to create a fill for all table cells using the color "light red" and a 50% transparency.

JavaScript

```

uniformCellProperties: {
  fill: {
    color: "lightred",
    transparency: 0.5
  }
}

```

```
    },  
  }  
}
```

Borders

Use the [TableCellProperties.borders](#) object to define borders for cells in the table. The following code sample shows how to set the borders of a cell in row 1 by column 1 to a red border with weight 3.

JavaScript

```
const columnCount = 3;  
const rowCount = 3;  
// Compact syntax to create a 2D array filled with empty and distinct objects.  
const specificCellProperties = Array(rowCount).fill(undefined).map(_ =>  
  Array(columnCount).fill(undefined).map(_ => ({})));  
const options: PowerPoint.TableAddOptions = {  
  values: [  
    ["1", "2", "3"],  
    ["4", "5", "6"],  
    ["7", "8", "9"]  
  ],  
  uniformCellProperties: {  
    fill: {  
      color: "lightcyan",  
      transparency: 0.5  
    },  
  },  
  specificCellProperties  
};  
options.specificCellProperties[1][1] = {  
  font: {  
    color: "red",  
    name: "Arial",  
    size: 50,  
    allCaps: true,  
    italic: true  
  },  
  borders: {  
    bottom: {  
      color: "red",  
      weight: 3  
    },  
    left: {  
      color: "red",  
      weight: 3  
    },  
    right: {  
      color: "red",  
      weight: 3  
    },  
  },  
};
```



```

        top: {
          color: "red",
          weight: 3
        }
      }
    };
    await insertTableOnCurrentSlide(rowCount, columnCount, options);

```

Horizontal and vertical alignment

Use the [TableCellProperties.horizontalAlignment](#) property to control text alignment in a cell. The following example shows how to set horizontal alignment to left, right, and center for three cells in a table. For a list of all alignment options, see the [ParagraphHorizontalAlignment](#) enum.

JavaScript

```

const rowCount = 3;
const columnCount = 3;
// Compact syntax to create a 2D array filled with empty and distinct objects.
const specificCellProperties = Array(rowCount).fill("").map(_ =>
Array(columnCount).fill("").map(_ => ({})));
const options: PowerPoint.TableAddOptions = {
  values: [
    ["Left aligned, top", "\n\n", ""],
    ["Centered", "\n\n", ""],
    ["Right aligned, bottom", "\n\n", ""]
  ],
  uniformCellProperties: {
    fill: { color: "lightblue" },
    borders: {
      bottom: {
        color: "black",
        weight: 3
      },
      left: {
        color: "black",
        weight: 3
      },
      right: {
        color: "black",
        weight: 3
      },
      top: {
        color: "black",
        weight: 3
      }
    }
  },
  specificCellProperties // Array values are empty objects at this point.
};

```

```
options.specificCellProperties[0][0] = {
    horizontalAlignment: PowerPoint.ParagraphHorizontalAlignment.left,
    verticalAlignment: 0 //PowerPoint.TextVerticalAlignment.top
};
options.specificCellProperties[1][0] = {
    horizontalAlignment: PowerPoint.ParagraphHorizontalAlignment.center,
    verticalAlignment: 1 //PowerPoint.TextVerticalAlignment.middle
};
options.specificCellProperties[2][0] = {
    horizontalAlignment: PowerPoint.ParagraphHorizontalAlignment.right,
    verticalAlignment: 2 //PowerPoint.TextVerticalAlignment.bottom
};
await insertTableOnCurrentSlide(3, 3, options);
```

The previous sample creates a table with left/top, centered, and right/bottom text alignment as shown in the following image.

Left aligned, top		
Centered		
Right aligned, bottom		

Specify row and column widths

Specify row and column widths using the [TableAddOptions.rows](#) and [TableAddOptions.columns](#) properties. The `rows` property is an array of [TableRowProperties](#) that you use to set each row's [rowHeight](#) property. Similarly, the `columns` property is an array of [TableColumnProperties](#) you use to set each column's [columnWidth](#) property. The width or height is set in points.

The height or width that you set may not be honored by PowerPoint if it needs to fit the text. For example, if the text is too wide for a column, PowerPoint will increase the row height so that it can wrap the text to the next line. Similarly, the column width will increase if the specified size is smaller than a single character in the specified font size.

The following code example shows how to set row height and column width for a new table. Note that the rows and columns properties must be set to an array of objects equal to their count.

```

const columnCount = 3;
const rowCount = 3;
const options: PowerPoint.TableAddOptions = {
  values: [
    ["Width 72pt", "Width 244pt", "Width 100pt"],
    ["", "", ""],
    ["", "^\\n\\nHeight 200 pt\\n\\nv", ""]
  ],
  // Initialize columns with an array of empty objects for each column.
  columns: Array(columnCount).fill("").map(_ => ({})),
  rows: Array(columnCount).fill("").map(_ => ({})),
  uniformCellProperties: {
    fill: { color: "lightcyan" },
    horizontalAlignment: PowerPoint.ParagraphHorizontalAlignment.center,
    verticalAlignment: 1, //PowerPoint.TextVerticalAlignment.middle
    borders: {
      bottom: {
        color: "black",
        weight: 3
      },
      left: {
        color: "black",
        weight: 3
      },
      right: {
        color: "black",
        weight: 3
      },
      top: {
        color: "black",
        weight: 3
      }
    }
  }
};
options.columns[0].columnWidth = 72;
options.columns[1].columnWidth = 244;
options.columns[2].columnWidth = 100;
options.rows[2].rowHeight = 200;
await insertTableOnCurrentSlide(rowCount, columnCount, options);

```

The previous sample creates a table with three custom column widths, and one custom row height, as shown in the following image.

Width 72pt	Width 244pt	Width 100pt
	^ Height 200 pt v	

Specify merged areas

A merged area is two or more cells combined so that they share a single value and format. In appearance the merged area spans multiple rows or columns. A merged area is indexed by its upper left table cell location (row, column) when setting its value or format. The upper left cell of the merged area is always used to set the value and formatting. All other cells in the merged area must be empty strings with no formatting applied.

To specify a merged area, provide the upper left location where the area starts (row, column) and the length of the area in rows and columns. The following diagram shows an example of these values for a merged area that is 3 rows by 2 columns in size. Note that merged areas can't overlap with each other.

Use the [TableAddOptions.mergedAreas](#) property to specify one or more merged areas. The following code sample shows how to create a table with two merged areas. About the code sample, note the following:

- The values property must only specify the value for the upper left corner of the merged area. All other cell values in the merged area must specify empty strings ("").
- Each merged area must specify the upper left corner location (row, column) and the length in cells of the merged area in terms of row count and column count.

JavaScript

```
const rowCount = 3;
const columnCount = 4;
```

```

// Compact syntax to create a 2D array filled with empty and distinct objects.
const specificCellProperties = Array(rowCount).fill("").map(_ =>
Array(columnCount).fill("").map(_ => ({})));
const options: PowerPoint.TableAddOptions = {
  values: [
    ["1", "This is a merged cell", "", "4"],
    ["5", "6", "This is also a merged cell", "8"],
    ["9", "10", "", "12"]
  ],
  uniformCellProperties: {
    fill: { color: "darkslateblue" },
    font: { bold: true, color: "white" },
    borders: {
      bottom: {
        color: "black",
        weight: 3
      },
      left: {
        color: "black",
        weight: 3
      },
      right: {
        color: "black",
        weight: 3
      },
      top: {
        color: "black",
        weight: 3
      }
    }
  },
  mergedAreas: [{ rowIndex: 0, columnIndex: 1, rowCount: 1, columnCount: 2 },
  { rowIndex: 1, columnIndex: 2, rowCount: 2, columnCount: 1 }
],
  specificCellProperties // Array values are empty objects at this point.
};
// Set fill color for specific cell at row 1, column 1.
options.specificCellProperties[1][1] = {
  fill: { color: "black" }
};
await insertTableOnCurrentSlide(rowCount, columnCount, options);

```

The previous sample creates a table with two merged areas as shown in the following image.

1	This is a merged cell		4
5	6	This is also a merged cell	8
9	10		12

Get and set table cell values

After a table is created you can get or set string values in the cells. Note that this is the only part of a table you can change. You can't change borders, fonts, widths, or other cell properties. If you need to update a table, delete it and recreate it. The following code sample shows how to find an existing table and set a new value for a cell in the table.

JavaScript

```
await PowerPoint.run(async (context) => {
    // Load shapes.
    const shapes = context.presentation.getSelectedSlides().getItemAt(0).shapes;
    shapes.load("items");
    await context.sync();
    // Find the first shape of type table.
    const shape = shapes.items.find((shape) => shape.type ===
PowerPoint.ShapeType.table)
    const table = shape.getTable();
    table.load("values");
    await context.sync();
    // Set the value of the specified table cell.
    let values = table.values;
    values[1][1] = "A new value";
    table.values = values;
    await context.sync();
});
```

You can also get the following read-only properties from the table.

- **rowCount**
- **columnCount**

The following sample shows how to get the table properties and log them to the console. The sample also shows how to get the merged areas in the table.

JavaScript

```
await PowerPoint.run(async (context) => {
    // Load shapes.
    const shapes = context.presentation.getSelectedSlides().getItemAt(0).shapes;
    shapes.load("items");
    await context.sync();
    // Find the first shape of type table.
    const shape = shapes.items.find((shape) => shape.type ===
PowerPoint.ShapeType.table)
    const table = shape.getTable();
    // Load row and column counts.
    table.load("rowCount, columnCount");
    // Load the merged areas.
    const mergedAreas = table.getMergedAreas();
    mergedAreas.load("items");
    await context.sync();
});
```

```
// Log the table properties.  
console.log(mergedAreas);  
console.log(table.rowCount);  
console.log(table.columnCount);  
});
```

Project add-ins documentation

With Project add-ins, you can use familiar web technologies such as HTML, CSS, and JavaScript to build a solution that can run in Project on Windows. Learn how to build, test, debug, and publish Project add-ins.

About Project add-ins

OVERVIEW

[What are Project add-ins?](#)

QUICKSTART

[Build your first Project add-in](#)

HOW-TO GUIDE

[Test and debug a Project add-in](#)

[Deploy and publish a Project add-in](#)

Key Office Add-ins concepts

OVERVIEW

[Office Add-ins platform overview](#)

GET STARTED

[Core concepts for Office Add-ins](#)

[Design Office Add-ins](#)

[Develop Office Add-ins](#)

Resources

REFERENCE

[Ask questions](#) ↗

[Request features](#) ↗

[Report issues](#) ↗

[Join Office Add-ins community call](#)

[Office Add-ins additional resources](#)

[Download samples](#)