Word add-ins documentation

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

About Word add-ins

OVERVIEW

What are Word add-ins?

QUICKSTART

Build your first Word add-in

Explore APIs with Script Lab

HOW-TO GUIDE

Use the Word JavaScript API to interact with document content and metadata

Test and debug a Word add-in

Deploy and publish a Word add-in

SAMPLE

Import a Word document template with a Word add-in

Manage citations through your Word add-in

Key Office Add-ins concepts

OVERVIEW

Office Add-ins platform overview

CONCEPT

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

Word add-ins overview

Article • 05/29/2025

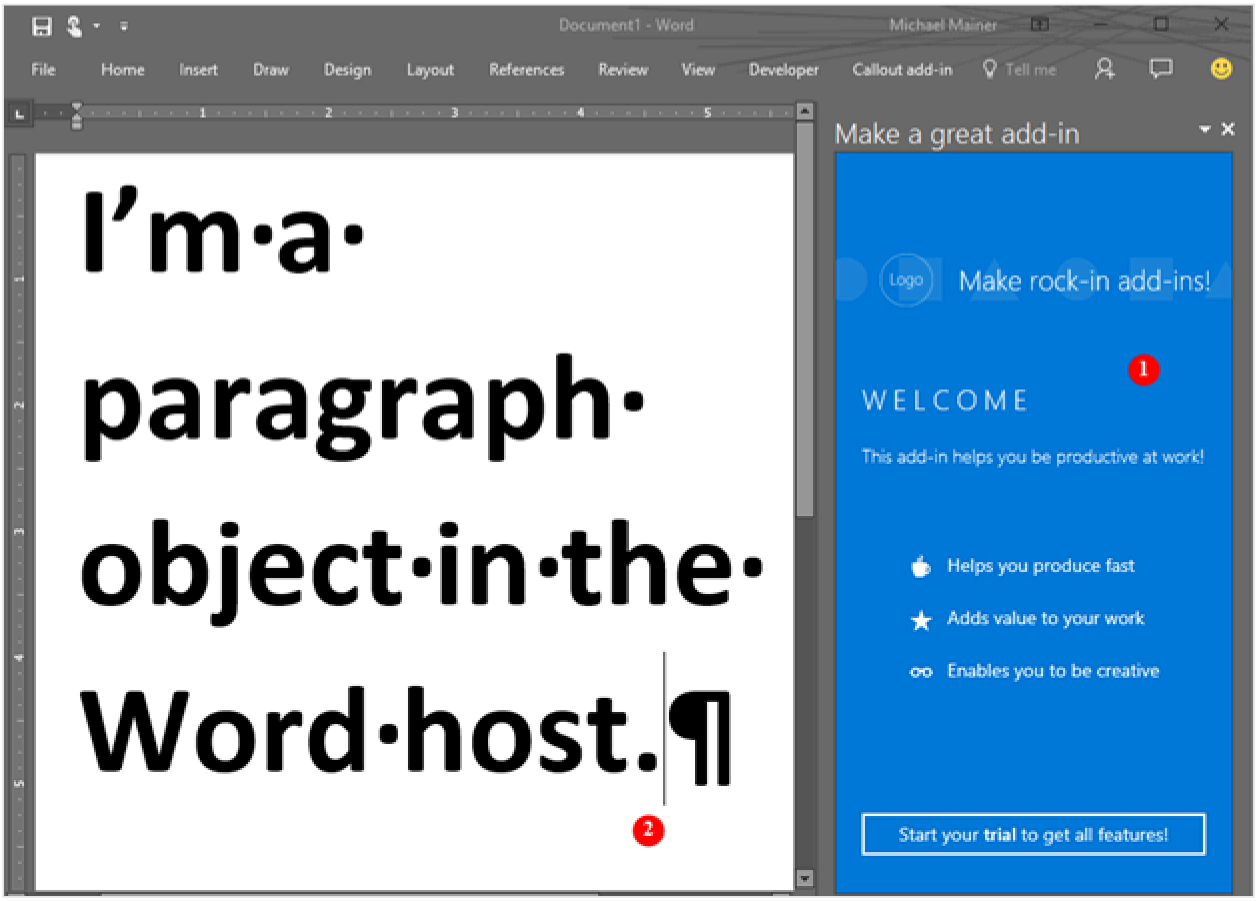
Do you want to create a solution that extends the functionality of Word? For example, one that involves automated document assembly? Or a solution that binds to and accesses data in a Word document from other data sources? You can use the Office Add-ins platform, which includes the Word JavaScript API and the Office JavaScript API, to extend Word clients running on the web, on a Windows desktop, or on a Mac.

Word add-ins are one of the many development options that you have on the Office Add-ins platform. You can use add-in commands to extend the Word UI and launch task panes that run JavaScript that interacts with the content in a Word document. Any code that you can run in a browser can run in a Word add-in. Add-ins that interact with content in a Word document create requests to act on Word objects and synchronize object state.

Note

If you plan to publish your add-in to AppSource and make it available within the Office experience, make sure that you conform to the Commercial marketplace certification policies. For example, to pass validation, your add-in must work across all platforms that support the methods that you define (for more information, see section 1120.3 and the Office Add-in application and availability page).

The following figure shows an example of a Word add-in that runs in a task pane.



The Word add-in can do the following:

1. Send requests to the Word document.
2. Use JavaScript to access the paragraph object and update, delete, or move the paragraph.

For example, the following code shows how to append a new sentence to the first paragraph.

JavaScript

```javascript
await Word.run(async (context) => {
  const paragraphs = context.document.body.paragraphs;
  paragraphs.load();
  await context.sync();

  paragraphs.items[0].insertText(' New sentence in the paragraph.',
                                 Word.InsertLocation.end);
  await context.sync();
});
```

You can use any web server technology to host your Word add-in, such as ASP.NET, NodeJS, or Python. Use your favorite client-side framework—Ember, Backbone, Angular, React—or stick with plain JavaScript to develop your solution. You can also use services like Microsoft Entra and Microsoft Azure to authenticate and host your application respectively.

The Word JavaScript APIs give your application access to the objects and metadata found in a Word document. You can use these APIs to create add-ins that target the following clients.

- Word on the web
- Word 2016 or later on Windows
- Word on Mac
- Word on iPad

Write your add-in once, and it will run in all supported versions of Word across multiple platforms. For details, see Office client application and platform availability for Office Add-ins.

JavaScript APIs for Word

You can use two sets of JavaScript APIs to interact with the objects and metadata in a Word document.

The first is the Word JavaScript API. This is an application-specific API model that was introduced with Word 2016. It's a strongly-typed object model that you can use to create Word add-ins that target Word 2016 and later on Windows and on Mac. This object model uses promises and provides access to Word-specific objects like body, content controls, inline pictures, and paragraphs. The Word JavaScript API includes TypeScript definitions and vsdoc files so that you can get code hints in your IDE.

The second is the Common API, which was introduced in Office 2013. Many of the objects in the Common API can be used in add-ins hosted by two or more Office clients. This API uses callbacks extensively.

Currently, all Word clients support Word JavaScript API and the shared Office JavaScript API. For details about supported clients, see Office client application and platform availability for Office Add-ins.

We recommend that you start with the Word JavaScript API because the object model is easier to use. Use the Word JavaScript API if you need to access the objects in a Word document.

Use the shared Office JavaScript API when you need to do any of the following:

- Perform initialize actions for the application.
- Check the supported requirement set.
- Access metadata, settings, and environmental information for the document.
- Bind to sections in a document and capture events.
- Open a dialog box.

Next steps

Ready to create your first Word add-in? See Build your first Word add-in. Use the add-in manifest to describe where your add-in is hosted, how it's displayed, and define permissions and other information.

To learn more about how to design a world-class Word add-in that creates a compelling experience for your users, see Design guidelines and Best practices.

After you develop your add-in, you can publish it to a network share, an app catalog, or AppSource.

See also

- Developing Office Add-ins
- Learn about the Microsoft 365 Developer Program
- Office Add-ins platform overview
- Word JavaScript API reference

Build your first Word task pane add-in

Article • 12/19/2024

In this article, you'll walk through the process of building a Word task pane add-in. You'll use either the Office Add-ins Development Kit or the Yeoman generator to create your Office Add-in. Select the tab for the one you'd like to use and then follow the instructions to create your add-in and test it locally. If you'd like to create the add-in project within Visual Studio Code, we recommend the Office Add-ins Development Kit.

Office Add-ins Development Kit

Prerequisites

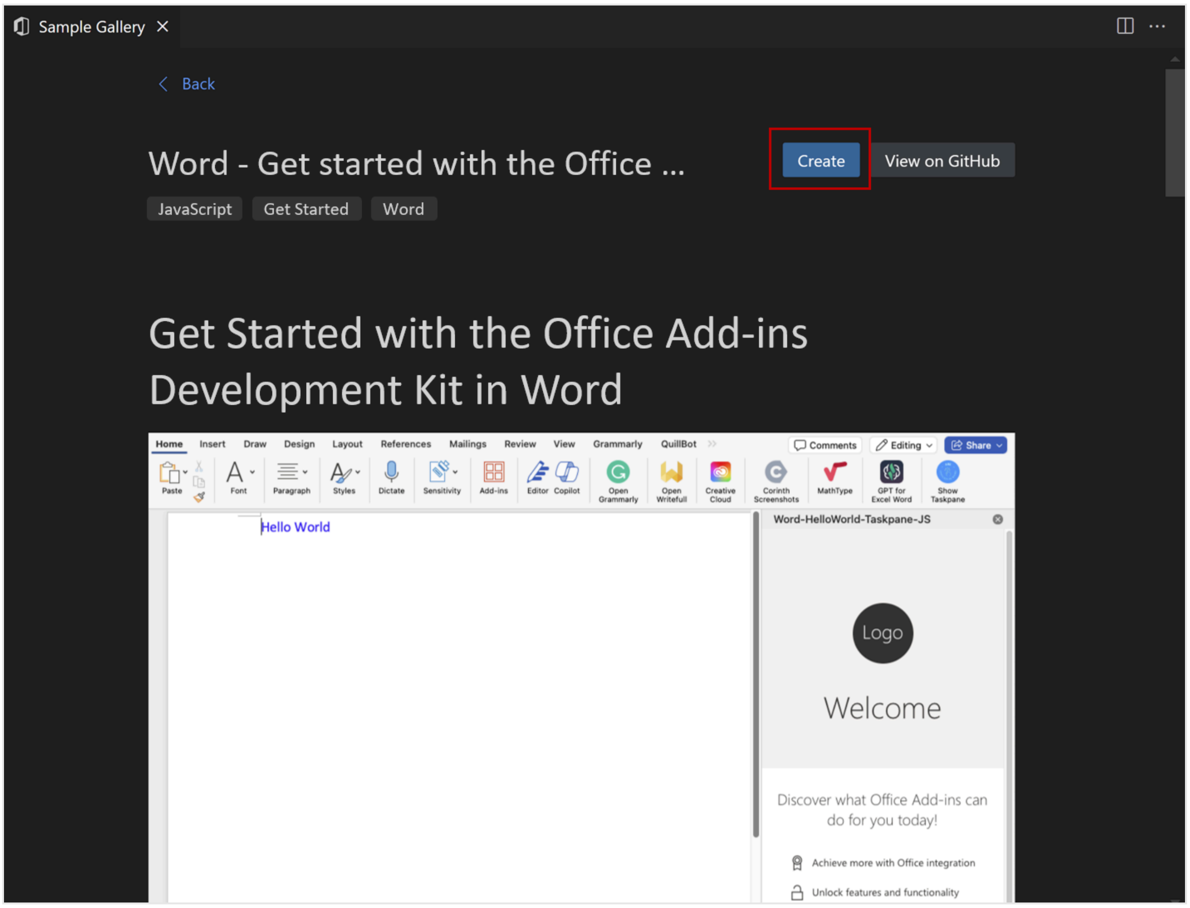
- Download and install Visual Studio Code.
- Node.js (the latest LTS version). Visit the Node.js site to download and install the right version for your operating system. To verify if you've already installed these tools, run the commands `node -v` and `npm -v` in your terminal.
- Office connected to a Microsoft 365 subscription. You might qualify for a Microsoft 365 E5 developer subscription through the Microsoft 365 Developer Program, see FAQ for details. Alternatively, you can sign up for a 1-month free trial or purchase a Microsoft 365 plan.

Create the add-in project

Click the following button to create an add-in project using the Office Add-ins Development Kit for Visual Studio Code. You'll be prompted to install the extension if don't already have it. A page that contains the project description will open in Visual Studio Code.

Create an add-in in Visual Studio Code

In the prompted page, select **Create** to create the add-in project. In the **Workspace folder** dialog that opens, select the folder where you want to create the project.


Sample Gallery
Back
Word - Get started with the Office ...
Create
View on GitHub
JavaScript
Get Started
Word
Get Started with the Office Add-ins Development Kit in Word
Hello World
Word-HelloWorld-Taskpane-JS
Logo
Welcome
Discover what Office Add-ins can do for you today!
Achieve more with Office integration
Unlock features and functionality

The Office Add-ins Development Kit will create the project. It will then open the project in a *second* Visual Studio Code window. Close the original Visual Studio Code window.

ⓘ Note

If you use VSCode Insiders, or you have problems opening the project page in VSCode, install the extension manually by following these steps, and find the sample in the sample gallery.

Explore the project

The add-in project that you've created with the Office Add-ins Development Kit 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.

1. The **./manifest.xml** or **./manifest.json** file in the root directory of the project defines the settings and capabilities of the add-in.
2. The **./src/taskpane/taskpane.html** file contains the HTML markup for the task pane.

3. The **./src/taskpane/taskpane.css** file contains the CSS that's applied to content in the task pane.
4. 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. Open the extension by selecting the Office Add-ins Development Kit icon in the **Activity Bar**.
2. Select **Preview Your Office Add-in (F5)**
3. In the Quick Pick menu, select the option **{Office Application} Desktop (Edge Chromium)**, where '{Office Application}' is the appropriate application, such as "Excel" or "Word". This will launch the add-in and debug the code.

The development kit checks that the prerequisites are met before debugging starts. Check the terminal for detailed information if there are issues with your environment. After this process, the Office desktop application launches and sideloads the add-in. Please note that the first time you run a project, it may make take a few minutes to install the dependencies. You'll need to install the certificate when prompted.

Stop testing your Office Add-in

Once you are finished testing and debugging the add-in, *always* close the add-in by following these steps. (Closing the Office application or web server window doesn't reliably deregister the add-in.)

1. Open the extension by selecting the Office Add-ins Development Kit icon in the **Activity Bar**.
2. Select **Stop Previewing Your Office Add-in**. This closes the web server and removes the add-in from the registry and cache.
3. Close the Office application window.

Troubleshooting

If you have problems running the add-in, take these steps.

- Close any open instances of Office.
- Close the previous web server started for the add-in with the **Stop Previewing Your Office Add-in** Office Add-ins Development Kit extension option.

The article Troubleshoot development errors with Office Add-ins contains solutions to common problems. If you're still having issues, create a GitHub issue and we'll help you.

For information on running the add-in on Office on the web, see Sideload Office Add-ins to Office on the web.

For information on debugging on older versions of Office, see Debug add-ins using developer tools in Microsoft Edge Legacy.

Tutorial: Create a Word task pane add-in

Article • 01/16/2025

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

- ✓ Inserts a range of text
- ✓ Formats text
- ✓ Replaces text and inserts text in various locations
- ✓ Inserts images, HTML, and tables
- ✓ Creates and updates content controls

Tip

If you've already completed the **Build your first Word task pane add-in** quick start, and want to use that project as a starting point for this tutorial, go directly to the **Insert a range of text** 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 your 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.

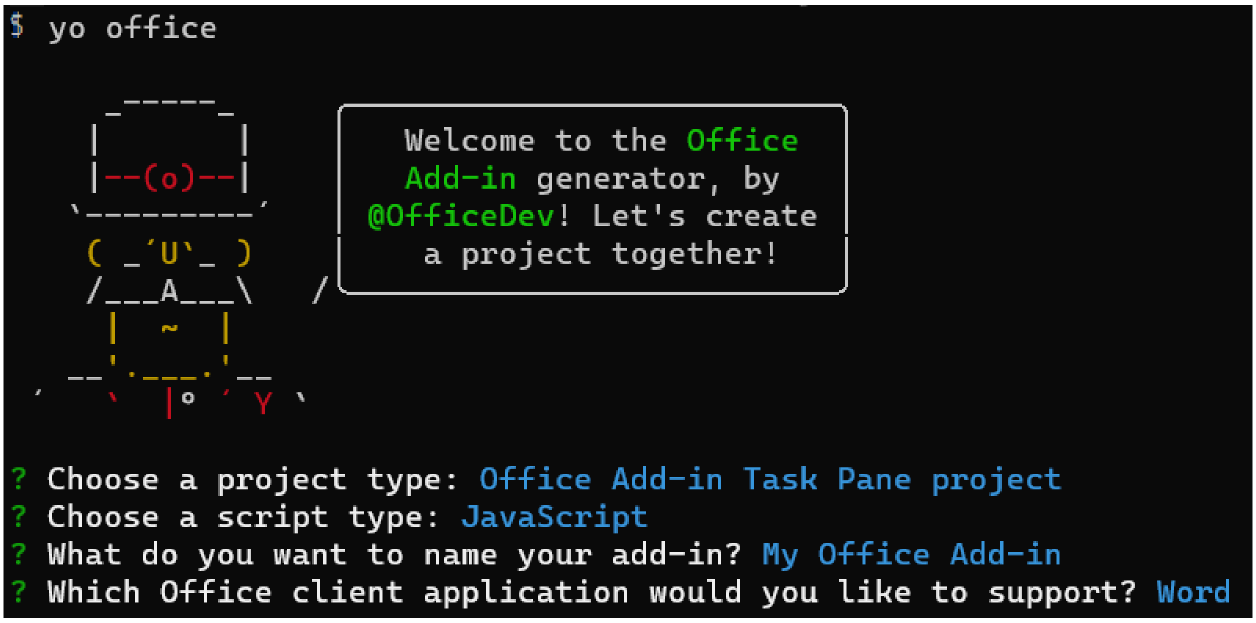
command line

```
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?** `Word`



```
$ yo office

     _-----_     ╭──────────────────────────╮
    |       |    │  Welcome to the Office   │
    |--(o)--|    │  Add-in generator, by    │
   `---------´   │ @OfficeDev! Let's create │
    ( _´U`_ )    │  a project together!     │
    /___A___\   /╰──────────────────────────╯
     |  ~  |
   __'.___.'__
 ´   `  |° ´ Y `

? 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? Word
```

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

Insert a range of text

In this step of the tutorial, you'll programmatically test that your add-in supports the user's current version of Word, and then insert a paragraph into the document.

Code the add-in

1. Open the project in your code editor.

2. Open the file **./src/taskpane/taskpane.html**. This file contains the HTML markup for the task pane.

3. Locate the `<main>` element and delete all lines that appear after the opening `<main>` tag and before the closing `</main>` tag.

4. Add the following markup immediately after the opening `<main>` tag.

   HTML

   ```html
   <button class="ms-Button" id="insert-paragraph">Insert
   Paragraph</button><br/><br/>
   ```

5. 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.

6. Remove all references to the `run` button and the `run()` function by doing the following:

   - Locate and delete the line `document.getElementById("run").onclick = run;`.

   - Locate and delete the entire `run()` function.

7. Within the `Office.onReady` function call, locate the line `if (info.host === Office.HostType.Word) {` and add the following code immediately after that line. Note:

   - This code adds an event handler for the `insert-paragraph` button.
   - The `insertParagraph` function is wrapped in a call to `tryCatch` (both functions will be added in the next step). This allows any errors generated by the Office JavaScript API layer to be handled separately from your service code.

   JavaScript

   ```
   // Assign event handlers and other initialization logic.
   document.getElementById("insert-paragraph").onclick = () =>
   tryCatch(insertParagraph);
   ```

8. Add the following functions to the end of the file. Note:

   - Your Word.js business logic will be added to the function passed to `Word.run`. This logic doesn't execute immediately. Instead, it's added to a queue of pending commands.

   - The `context.sync` method sends all queued commands to Word for execution.

   - The `tryCatch` function will be used by all the functions interacting with the workbook from the task pane. Catching Office JavaScript errors in this fashion is a convenient way to generically handle uncaught errors.

   JavaScript

   ```
   async function insertParagraph() {
       await Word.run(async (context) => {

           // TODO1: Queue commands to insert a paragraph into the
   document.

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

```
}

/** Default helper for invoking an action and handling errors. */
async function tryCatch(callback) {
    try {
        await callback();
    } catch (error) {
        // Note: In a production add-in, you'd want to notify the user
through your add-in's UI.
        console.error(error);
    }
}
```

9. Within the `insertParagraph()` function, replace `TODO1` with the following code. Note:

   - The first parameter to the `insertParagraph` method is the text for the new paragraph.

   - The second parameter is the location within the body where the paragraph will be inserted. Other options for insert paragraph, when the parent object is the body, are "End" and "Replace".

   JavaScript

   ```javascript
   const docBody = context.document.body;
   docBody.insertParagraph("Office has several versions, including Office
   2016, Microsoft 365 subscription, and Office on the web.",
                           Word.InsertLocation.start);
   ```

10. Save all your changes to the project.

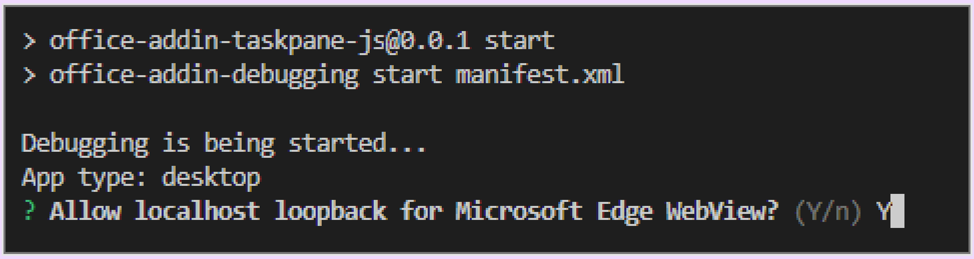
Test the add-in

1. 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 in the root directory of your project before proceeding. When you run this command, the local web server starts.

command line

```
npm run dev-server
```

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

command line

```
npm start
```

- To test your add-in in Word on the web, 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 Word 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

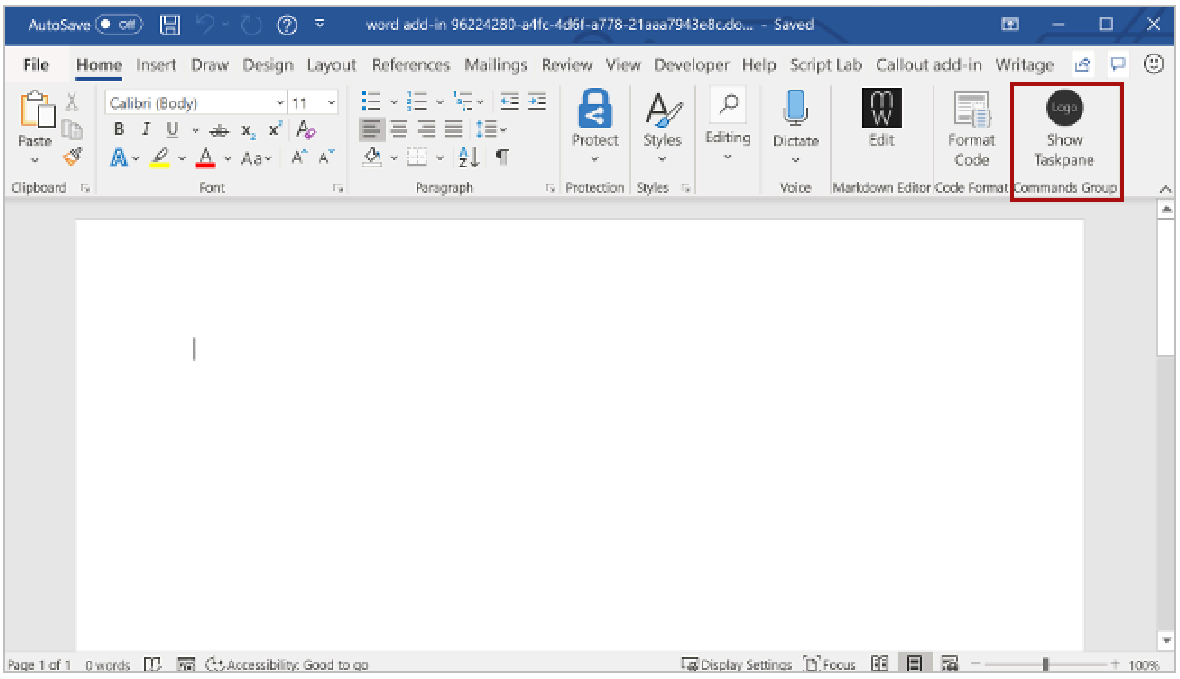
```
npm run start -- web --document {url}
```

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!jkcH7spkM4EGgcZUgqthk4IK3NOypVw?e=Z6G1qp`
- `npm run start -- web --document https://contoso-my.sharepoint-df.com/:t:/p/user/EQda453DNTpFnl1bFPhOVR0BwlrzetbXvnaRYii2lDr_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.

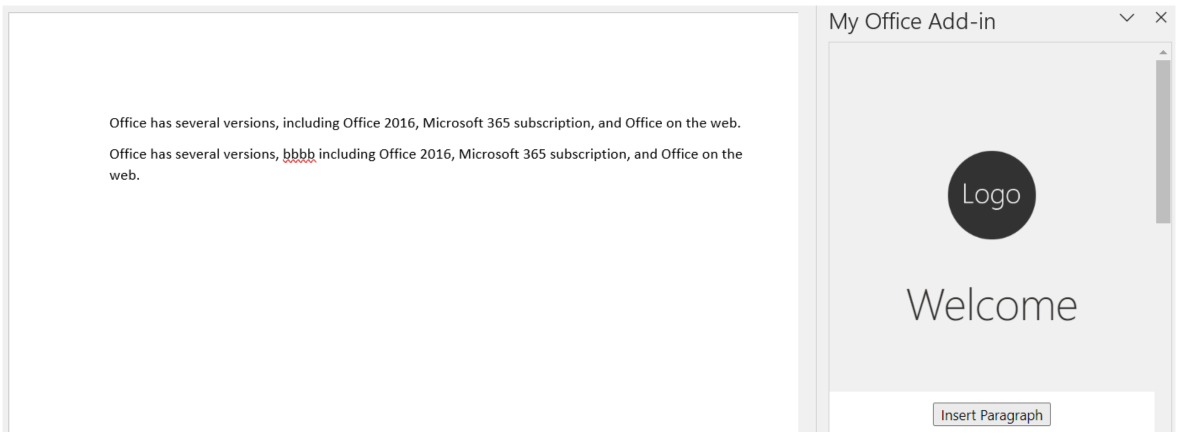
2. In Word, if the "My Office Add-in" task pane isn't already open, choose the **Home** tab, and then choose the **Show Taskpane** button on the ribbon to open the add-in task pane.



3. In the task pane, choose the **Insert Paragraph** button.

4. Make a change in the paragraph.

5. Choose the **Insert Paragraph** button again. Note that the new paragraph appears above the previous one because the `insertParagraph` method is inserting at the start of the document's body.



Office has several versions, including Office 2016, Microsoft 365 subscription, and Office on the web.

Office has several versions, bbbb including Office 2016, Microsoft 365 subscription, and Office on the web.

6. 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.

Format text

In this step of the tutorial, you'll apply a built-in style to text, apply a custom style to text, and change the font of text.

Apply a built-in style to text

1. Open the file **./src/taskpane/taskpane.html**.

2. Locate the `<button>` element for the `insert-paragraph` button, and add the following markup after that line.

   HTML

   ```html
   <button class="ms-Button" id="apply-style">Apply Style</button><br/>
   <br/>
   ```

3. Open the file **./src/taskpane/taskpane.js**.

4. Within the `Office.onReady` function call, locate the line that assigns a click handler to the `insert-paragraph` button, and add the following code after that line.

   JavaScript

   ```javascript
   document.getElementById("apply-style").onclick = () =>
   tryCatch(applyStyle);
   ```

5. Add the following function to the end of the file.

   JavaScript

   ```javascript
   async function applyStyle() {
       await Word.run(async (context) => {

           // TODO1: Queue commands to style text.

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

6. Within the `applyStyle()` function, replace `TODO1` with the following code. Note that the code applies a style to a paragraph, but styles can also be applied to ranges of text.

   JavaScript

```
const firstParagraph = context.document.body.paragraphs.getFirst();
firstParagraph.styleBuiltIn = Word.Style.intenseReference;
```

Apply a custom style to text

1. Open the file **./src/taskpane/taskpane.html**.

2. Locate the `<button>` element for the `apply-style` button, and add the following markup after that line.

   HTML

   ```html
   <button class="ms-Button" id="apply-custom-style">Apply Custom
   Style</button><br/><br/>
   ```

3. Open the file **./src/taskpane/taskpane.js**.

4. Within the `Office.onReady` function call, locate the line that assigns a click handler to the `apply-style` button, and add the following code after that line.

   JavaScript

   ```javascript
   document.getElementById("apply-custom-style").onclick = () =>
   tryCatch(applyCustomStyle);
   ```

5. Add the following function to the end of the file.

   JavaScript

   ```javascript
   async function applyCustomStyle() {
       await Word.run(async (context) => {

           // TODO1: Queue commands to apply the custom style.

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

6. Within the `applyCustomStyle()` function, replace `TODO1` with the following code. Note that the code applies a custom style that does not exist yet. You'll create a style with the name **MyCustomStyle** in the Test the add-in step.

   JavaScript

```
const lastParagraph = context.document.body.paragraphs.getLast();
lastParagraph.style = "MyCustomStyle";
```

7. Save all your changes to the project.

Change the font of text

1. Open the file **./src/taskpane/taskpane.html**.

2. Locate the `<button>` element for the `apply-custom-style` button, and add the following markup after that line.

   HTML

   ```html
   <button class="ms-Button" id="change-font">Change Font</button><br/>
   <br/>
   ```

3. Open the file **./src/taskpane/taskpane.js**.

4. Within the `Office.onReady` function call, locate the line that assigns a click handler to the `apply-custom-style` button, and add the following code after that line.

   JavaScript

   ```javascript
   document.getElementById("change-font").onclick = () =>
   tryCatch(changeFont);
   ```

5. Add the following function to the end of the file.

   JavaScript

   ```javascript
   async function changeFont() {
       await Word.run(async (context) => {

           // TODO1: Queue commands to apply a different font.

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

6. Within the `changeFont()` function, replace `TODO1` with the following code. Note that the code gets a reference to the second paragraph by using the `ParagraphCollection.getFirst` method chained to the `Paragraph.getNext` method.

JavaScript

```
const secondParagraph =
context.document.body.paragraphs.getFirst().getNext();
secondParagraph.font.set({
        name: "Courier New",
        bold: true,
        size: 18
    });
```

7. Save all your changes to the project.

Test the add-in

1. If the local web server is already running and your add-in is already loaded in Word, proceed to step 2. Otherwise, start the local web server and sideload your add-in.

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

     command line

     ```
     npm start
     ```

   - To test your add-in in Word on the web, 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 Word 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!jkcH7spkM4EGgcZUgqthk4IK3NOypVw?e=Z6G1qp`
  - `npm run start -- web --document https://contoso-my.sharepoint-df.com/:t:/p/user/EQda453DNTpFnl1bFPhOVR0BwlrzetbXvnaRYii2lDr_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 Word, go to the **Home** tab and choose the **Show Taskpane** button on the ribbon to open it.

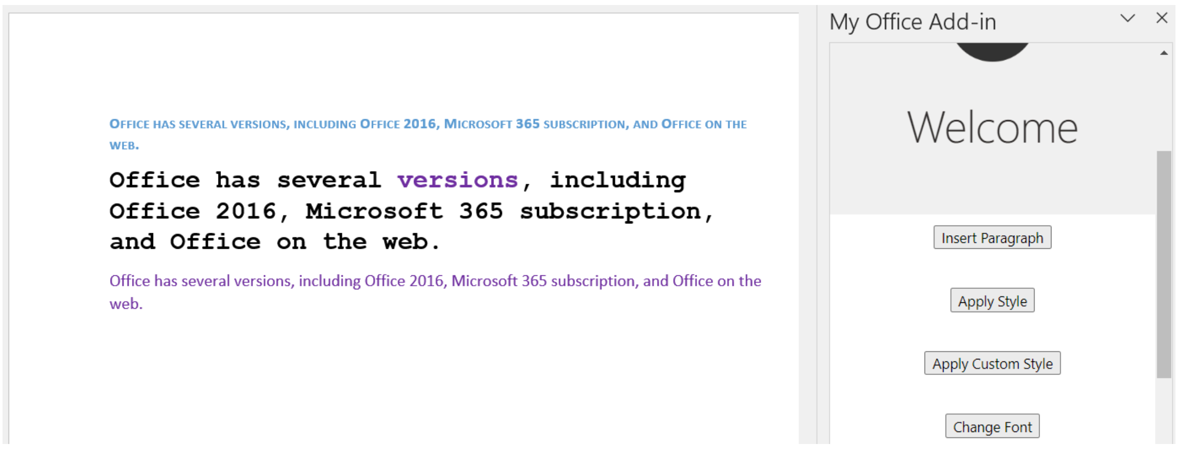
3. Be sure there are at least three paragraphs in the document. You can choose the **Insert Paragraph** button three times. *Check carefully that there's no blank paragraph at the end of the document. If there is, delete it.*

4. In Word, create a custom style named "MyCustomStyle". It can have any formatting that you want.

5. Choose the **Apply Style** button. The first paragraph will be styled with the built-in style **Intense Reference**.

6. Choose the **Apply Custom Style** button. The last paragraph will be styled with your custom style. (If nothing seems to happen, the last paragraph might be blank. If so, add some text to it.)

7. Choose the **Change Font** button. The font of the second paragraph changes to 18 pt., bold, Courier New.


OFFICE HAS SEVERAL VERSIONS, INCLUDING OFFICE 2016, MICROSOFT 365 SUBSCRIPTION, AND OFFICE ON THE WEB.

Office has several versions, including Office 2016, Microsoft 365 subscription, and Office on the web.

Office has several versions, including Office 2016, Microsoft 365 subscription, and Office on the web.

Replace text and insert text

In this step of the tutorial, you'll add text inside and outside of selected ranges of text, and replace the text of a selected range.

Add text inside a range

1. Open the file **./src/taskpane/taskpane.html**.

2. Locate the `<button>` element for the `change-font` button, and add the following markup after that line.

   HTML

   ```html
   <button class="ms-Button" id="insert-text-into-range">Insert
   Abbreviation</button><br/><br/>
   ```

3. Open the file **./src/taskpane/taskpane.js**.

4. Within the `Office.onReady` function call, locate the line that assigns a click handler to the `change-font` button, and add the following code after that line.

   JavaScript

   ```javascript
   document.getElementById("insert-text-into-range").onclick = () =>
   tryCatch(insertTextIntoRange);
   ```

5. Add the following function to the end of the file.

   JavaScript

   ```javascript
   async function insertTextIntoRange() {
       await Word.run(async (context) => {

           // TODO1: Queue commands to insert text into a selected range.

           // TODO2: Load the text of the range and sync so that the
           //        current range text can be read.

           // TODO3: Queue commands to repeat the text of the original
           //        range at the end of the document.

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

6. Within the `insertTextIntoRange()` function, replace `TODO1` with the following code. Note:

   - The function is intended to insert the abbreviation ["(M365)"] into the end of the Range whose text is "Microsoft 365". It makes a simplifying assumption that the string is present and the user has selected it.

   - The first parameter of the `Range.insertText` method is the string to insert into the `Range` object.

   - The second parameter specifies where in the range the additional text should be inserted. Besides "End", the other possible options are "Start", "Before", "After", and "Replace".

   - The difference between "End" and "After" is that "End" inserts the new text inside the end of the existing range, but "After" creates a new range with the string and inserts the new range after the existing range. Similarly, "Start" inserts text inside the beginning of the existing range and "Before" inserts a new range. "Replace" replaces the text of the existing range with the string in the first parameter.

   - You saw in an earlier stage of the tutorial that the `insert*` methods of the body object don't have the "Before" and "After" options. This is because you can't put content outside of the document's body.

   JavaScript

   ```javascript
   const doc = context.document;
   const originalRange = doc.getSelection();
   originalRange.insertText(" (M365)", Word.InsertLocation.end);
   ```

7. We'll skip over `TODO2` until the next section. Within the `insertTextIntoRange()` function, replace `TODO3` with the following code. This code is similar to the code you created in the first stage of the tutorial, except that now you are inserting a new paragraph at the end of the document instead of at the start. This new paragraph will demonstrate that the new text is now part of the original range.

   JavaScript

   ```javascript
   doc.body.insertParagraph("Original range: " + originalRange.text,
   Word.InsertLocation.end);
   ```

Add code to fetch document properties into the task pane's script objects

In all previous functions in this tutorial, you queued commands to *write* to the Office document. Each function ended with a call to the `context.sync()` method which sends the queued commands to the document to be executed. But the code you added in the last step calls the `originalRange.text` property, and this is a significant difference from the earlier functions you wrote, because the `originalRange` object is only a proxy object that exists in your task pane's script. It doesn't know what the actual text of the range in the document is, so its `text` property can't have a real value. It's necessary to first fetch the text value of the range from the document and use it to set the value of `originalRange.text`. Only then can `originalRange.text` be called without causing an exception to be thrown. This fetching process has three steps.

1. Queue a command to load (that is, fetch) the properties that your code needs to read.
2. Call the context object's `sync` method to send the queued command to the document for execution and return the requested information.
3. Because the `sync` method is asynchronous, ensure that it has completed before your code calls the properties that were fetched.

The following step must be completed whenever your code needs to *read* information from the Office document.

1. Within the `insertTextIntoRange()` function, replace `TODO2` with the following code.

   JavaScript

   ```javascript
   originalRange.load("text");
   await context.sync();
   ```

When you're done, the entire function should look like the following:

JavaScript

```javascript
async function insertTextIntoRange() {
    await Word.run(async (context) => {

        const doc = context.document;
        const originalRange = doc.getSelection();
        originalRange.insertText(" (M365)", Word.InsertLocation.end);

        originalRange.load("text");
```

```
        await context.sync();

        doc.body.insertParagraph("Original range: " + originalRange.text,
Word.InsertLocation.end);

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

Add text between ranges

1. Open the file **./src/taskpane/taskpane.html**.

2. Locate the `<button>` element for the `insert-text-into-range` button, and add the following markup after that line.

   HTML

   ```
   <button class="ms-Button" id="insert-text-outside-range">Add Version
   Info</button><br/><br/>
   ```

3. Open the file **./src/taskpane/taskpane.js**.

4. Within the `Office.onReady` function call, locate the line that assigns a click handler to the `insert-text-into-range` button, and add the following code after that line.

   JavaScript

   ```
   document.getElementById("insert-text-outside-range").onclick = () =>
   tryCatch(insertTextBeforeRange);
   ```

5. Add the following function to the end of the file.

   JavaScript

   ```
   async function insertTextBeforeRange() {
       await Word.run(async (context) => {

           // TODO1: Queue commands to insert a new range before the
           //        selected range.

           // TODO2: Load the text of the original range and sync so that
   the
           //        range text can be read and inserted.
   ```

```
    });
}
```

6. Within the `insertTextBeforeRange()` function, replace `TODO1` with the following code. Note:

   - The function is intended to add a range whose text is "Office 2019, " before the range with text "Microsoft 365". It makes an assumption that the string is present and the user has selected it.
   - The first parameter of the `Range.insertText` method is the string to add.
   - The second parameter specifies where in the range the additional text should be inserted. For more details about the location options, see the previous discussion of the `insertTextIntoRange` function.

JavaScript

```javascript
const doc = context.document;
const originalRange = doc.getSelection();
originalRange.insertText("Office 2019, ", Word.InsertLocation.before);
```

7. Within the `insertTextBeforeRange()` function, replace `TODO2` with the following code.

JavaScript

```javascript
originalRange.load("text");
await context.sync();

// TODO3: Queue commands to insert the original range as a
//        paragraph at the end of the document.

// TODO4: Make a final call of context.sync here and ensure
//        that it runs after the insertParagraph has been queued.
```

8. Replace `TODO3` with the following code. This new paragraph will demonstrate the fact that the new text is ***not*** part of the original selected range. The original range still has only the text it had when it was selected.

JavaScript

```javascript
doc.body.insertParagraph("Current text of original range: " +
originalRange.text, Word.InsertLocation.end);
```

9. Replace `TODO4` with the following code.

JavaScript

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

Replace the text of a range

1. Open the file **./src/taskpane/taskpane.html**.

2. Locate the `<button>` element for the `insert-text-outside-range` button, and add the following markup after that line.

HTML

```html
<button class="ms-Button" id="replace-text">Change Quantity
Term</button><br/><br/>
```

3. Open the file **./src/taskpane/taskpane.js**.

4. Within the `Office.onReady` function call, locate the line that assigns a click handler to the `insert-text-outside-range` button, and add the following code after that line.

JavaScript

```javascript
document.getElementById("replace-text").onclick = () =>
tryCatch(replaceText);
```

5. Add the following function to the end of the file.

JavaScript

```javascript
async function replaceText() {
    await Word.run(async (context) => {

        // TODO1: Queue commands to replace the text.

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

6. Within the `replaceText()` function, replace `TODO1` with the following code. Note that the function is intended to replace the string "several" with the string "many".

It makes a simplifying assumption that the string is present and the user has selected it.

JavaScript

```
const doc = context.document;
const originalRange = doc.getSelection();
originalRange.insertText("many", Word.InsertLocation.replace);
```

7. Save all your changes to the project.

Test the add-in

1. If the local web server is already running and your add-in is already loaded in Word, proceed to step 2. Otherwise, start the local web server and sideload your add-in.

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

     command line

     ```
     npm start
     ```

   - To test your add-in in Word on the web, 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 Word 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/EZGxP7ksiE5DuxvY638G798BpuhwluxCM` `fF1WZQj3VYhYQ?e=F4QM1R`

  - `npm run start -- web --document https://1drv.ms/x/s!jkcH7spkM4EGgcZUgqthk4IK3NOypVw?e=Z6G1qp`
  - `npm run start -- web --document https://contoso-my.sharepoint-df.com/:t:/p/user/EQda453DNTpFnl1bFPhOVR0BwlrzetbXvnaRYii2lDr_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 Word, go to the **Home** tab and choose the **Show Taskpane** button on the ribbon to open it.

3. In the task pane, choose the **Insert Paragraph** button to ensure that there's a paragraph at the start of the document.

4. Within the document, select the phrase "Microsoft 365 subscription". *Be careful not to include the preceding space or following comma in the selection.*

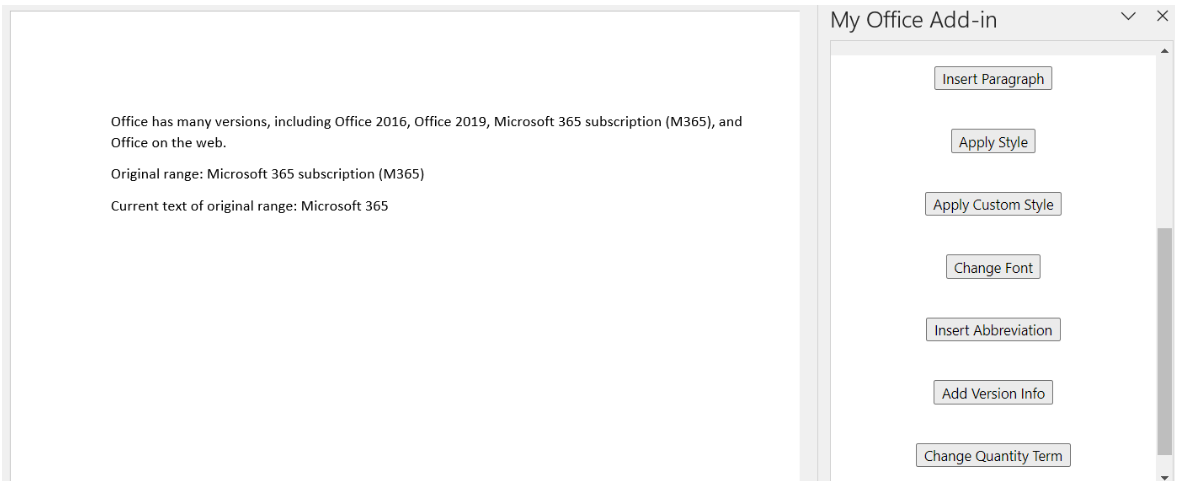
5. Choose the **Insert Abbreviation** button. Note that " (M365)" is added. Note also that at the bottom of the document a new paragraph is added with the entire expanded text because the new string was added to the existing range.

6. Within the document, select the phrase "Microsoft 365". *Be careful not to include the preceding or following space in the selection.*

7. Choose the **Add Version Info** button. Note that "Office 2019, " is inserted between "Office 2016" and "Microsoft 365". Note also that at the bottom of the document a new paragraph is added but it contains only the originally selected text because the new string became a new range rather than being added to the original range.

8. Within the document, select the word "several". *Be careful not to include the preceding or following space in the selection.*

9. Choose the **Change Quantity Term** button. Note that "many" replaces the selected text.



Office has many versions, including Office 2016, Office 2019, Microsoft 365 subscription (M365), and Office on the web.

Original range: Microsoft 365 subscription (M365)

Current text of original range: Microsoft 365

Insert images, HTML, and tables

In this step of the tutorial, you'll learn how to insert images, HTML, and tables into the document.

Define an image

Complete the following steps to define the image that you'll insert into the document in the next part of this tutorial.

1. In the root of the project, create a new file named **base64Image.js**.

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

JavaScript

```
export const base64Image =

"iVBORw0KGgoAAAANSUhEUgAAAZAAAAEFCAIAAABCdiZrAAAACXBIWXMAAAsSAAALEgHS3X
78AAAgAElEQVR42u2dzW9bV3rGn0w5wLBTRpSACAUDmDRowGoj1DdAtBA6suksZmtmV3Qj+
i8w3XUB00X3pv8CX68Gswq96aKLhI5bCKiM+gpVphIa1qQBcQbyQB/hTJlpOHUXlyEvD885
vLxfvCSfH7KIJVuUrnif+z7nPOd933v37h0IIWQe+BEvASGEgkUIIRQsQggFixBCKFiEEEEL
BIoRQsAghhIJFCCEULEIIBYsQQihYhBBCwSKEULAIIYSCRQghFCxCCAWLEEIoWIQQQsEihC
wQCV4CEgDdJvYM9C77f9x8gkyJV4UEznvs6U780rvAfgGdg5EPbr9CyuC1IbSEJGa8KopqB
WC/gI7Fa0MoWCROHJZw/l3lisITeBa8QoWCRyOk2JR9svdF+qvwnnQPsF+SrWSEjGa8KopqB
Cr0LNCo4rYkfb5s4vj/h33YOcFSWy59VlIsgIRQs4pHTGvYMdJvIjupOx5Ir0Tjtp5K/mTK
wXsSLq2hUWG0R93CXkKg9oL0+ldnFpil+yhlicIM06NA2cXgXySyuV7Fe5CUnFCziziyQO2qm
g8BIDUDWzVkUiPfHY8xOCGT77EWkH84FEZbx4DwOotbJpI5nj5CQWLTLMBj8vota8P2P8
KJWABIr2KpLwlmHpeHKff4BsmXxFQmhYBGlBxzoy7YljxOcfFAMottS6JH+4Xh69IhEgoW
cesBNdVQozLyd7whrdrGbSYdIqFgkQkecMD4ep09QB4I46v4tmbtGeK3QYdIKFhE7gEHjO/
odSzsfRzkS1+5h42q+MGOhf2CuPlIh0goWPSAogcccP2RJHI1riP+kQYdVK9Fh0goWPSAk8
2a5xCDG4zPJaWTxnvSIVKwKFj0gEq1go8QgxtUQQeNZtEhUrB4FZbaA9pIN+98hhcatbNp
qRoGgRKpdAhUrDIMnpAjVrpJSNApK/uRi7pEClYZIk84KDGQ+IBhhicMP6HRg1ycedVI6
RELBWl4POFCr8VWkszpe3o76G1aFs9ws+dMhUrDIInvAAeMB0ZBCDG6QBh2kgVI6RAoWX
PqBEI9+oQEtKgg3sNpUOkYJGF8oADAxDxgOIoUauXKIKOkxV99EhUrDIgnhAx0TfVfTDo
```

```
4JgOn3AegQKVhkvj2gjXRLLrIQgxtUQYdpNYsOkYJF5tUDarQg4hCDS1u3VZd83IOw0iFSs
MiceUCNWp3WYH0Wx59R6ls9W1c6RAoWmQ8PaCNdz55hiMEN4zsDNhMDpXSIFCwylx5Qo1a9
C3yVi69a2ajCWZ43NOkQKVgkph5wwHi+KQ4hBs9SC9+RMTpEChaJlwfUFylWEafP5uMKqII
OPv0sHSIFi8TFAzpLiXxF/KCbdetEGutFUSa6TXQsdKypv42UgZQhfrWOhbO6q8nPqqCD/z
U4OkQKFpm9B7SRbrTpQwzJHNaL/VHyiRVF0dfC2xpOzMnKlUgjW0amhGRW/ZM+w5sqzuqTN
Wtb9nKBZDLoEClYZGYe0EYaENWHGDaquHJv5CPnz/H9BToWkjmsFkTdOX0GS22p1ovYNEdU
r9vCeR3dJlIG1gojn2o8RKPiRX+D0iw6RAoWmYEH1HioiQZqq47VW32dalUlfi1fQf7ByEd
UQpMpYfOJ46UPcFweKaMSaWyaWL8z/Mibxzgqe3G4CC6pT4dIwSLReUCNWrkJMdjh8sMSuk
1d3bReRGb3hy97iS/SEl+5bQ0LqM4B9gvytaptC6kbwz++vD3ZG0r3EBDoWUg6RAoWCd0D9
isXReTKTYghZbhdUB/UYlKV2TSHitZtYc9QrqynDGy/GnGg+4XJr779ShJ0gNdAKR3i/PAj
XoIZe8BGBS+uhqtWAF4VXUWu3G//ORVqdVRiEumhWgFoVHT7gB1LnFAvVaJxYZJ+qx/XRuo
1X0+RFqzPsF/QFZuEgrVcHnDPCGbFylnajN/wAZZvqgpR8Iz0275tTvjnwl/4sORC6C9xWJ
LoYCKNrbpuR3Jazp/jxdUJmksoWIvvAfcLsD4LuLfn5hOJhWlVQ+lyNZDFcUl636GY5/Wpy
zo3FRZ+WBeT1JhpGDVlIMMbjYfYM3Ba4zuXgkUPGBD5B5Kl6LaJ4/uh/CCDTvDjW4ROxZm4
gj7+dwZLY24067AkF9OtesCaRYdIwaIHDIzMrmSzv2NNTgl4fLlSXw6kjs8pWN+FfHu3n8p
/xpSBjWrwL0eHSMGiB/TL+h1JnNJ+xTA6MawXh1ogTWA5S5tvLS8vMVUM6s1j+TKZEASjQ6
RgkVl6wH4pcUM+zs8qBq9WyRyMGozP+5J0/nzygrrLSkS4ONPmNg/vyr1npiQG9+kQKVhkB
h5woFbSI8EuQwxTkS1j2xoG0zsHeBVcRsl/RNMqyoMOG9WRjAUd4pzD4GhoHjDsMIEqchX4
8JuUgU1zJN+kSa4D+LnjHfXiqqsa5Oejb8J/fs9TAZjFtiXXvgADpaqXZsqUFRY94NRq1ag
ErFbrRWzVR9Tq9Jl0rWy75NncCf982n+o+sYCDJTSIVKw6AGnRhoQbZsBv3S+MlyxAtC7xP
F9WMUJDsi5M+gmVCWImpvolorOgXzTMPBAKR0iBWvuPWB4+4CiWj2Rz3MPcFSXHb90Nmawb
WDLRVZAc2pHZTkF2fWDKugQRqBUCvcQKVj0gI6qRxYQtfvGBIUdvHQ2fmk/VR7fk5Q5jr+2
fmfygrpTfM+fu8qa6lEFHcIIlGocolWkQwwcLrr79oBB9YRxg7SDXbDjJISue71LHJWnrno
+vRh+BX2Xq2QO06+Hf3TTXsYl43M3BhVcZFNjEyvIluUNvAgrrIX1gINqRdpvM0C1EhatbB
vowaM5neOVe/L2VX176/jip88CUysAhyV5SRheoFRSfV+i8RAvckH+XKyweBW8qNWeEelEP
1XkKqgQw3j/T3sxyNv6cSKNm02xA3KrOvLV1gq4Xh1u3vUusWcE7KESK7jZlHvSoDqU+q/4
CAUrItomWtUoRvup1KpRCWxb0KiNqFXvcoreWCem/ETh+ILRYJnvJzlxz+7wrt/l9qkuHUI
IrMk9bxaZEjIltl2mYMWDjoVWFae1sAouVeQq2LUYZwfRaVG1dR9PnKp802EpxG016TCOgZ
sOb6tk9RayZVZVFKwZ8cff4b/+Htcq8sd17wInJt5UA17SUqnVWR0vbwf5Qn5KgPO6bo0mU
0K2LJetbgtvqjgxQw8uqcbthDH+OrHS/5FV19MuJDXreoSCFQC9C3yxisQK8hVk1dteZ3W8
qQY2VFm68OF/emj0JNJ430DKQCKN3gU6FrrNSHf9VaMrfI68F+ynXVKpkhxndRyX0TlQzv4
hFKyABWuwMPGROWxiJ6kdmmibaJu+7gTpPRbgDbZsqJa9/T8AMrvIlnWx/m4Tx+XhY4yC5R
XGGjzRbeHlbd3ZsWQO+Qp2mth84nFtSBoQtS0M1cobqqCD50BpMovrj/Dpufyk1OBXZueKg
yq6KVjEI/bZMf3ef6aErTp2XiOzO8UtIe0gCuCoHMWm5MLWyJfK09HTdihdvwPjc+w0J4wv
bJv4KhfF2VIKFnHLm8f4KjfhkF0yh00TN5vYfDJ510wVED0qR7ENv7Sa5SZQmlhB/gF2XsO
oTdj+O6tjz8Dh3Tlbaow9XMNy/153rGGpDIJ+Ycv5bm6bcvVR5YaiPFCy8Kze6s+4lj4VpI
HS1Vv4sORqa09YrlL5fa5hUbBmLFiDd/am6Soi0LtAqzqyMK9Sq8BDDEQVdMBooDSxgvXih
AV14RfqxgBSsChYcREsmyv3lImtcU5raJs4q8sjV/MYYpgLrj9SxlP2C/iuiXxFl1EYL4GP
ym5/TRQsCla8BKu/3qFNbLl80a9yVKuwUIWzpmKQrnIPBcsrXHQPT+AucXzf70l91lahclT
2FV7tNmEV8fI2t24jI8FLEC52Ysv9wpbAtsVLGNNy2+VyFWGFNX+4SWyReYHpKgrWUuAmsU
XiDNNVFKwlsxJBLGyRGVh7LlfFAq5hzeTd38LL27oo0ABpnykSIG766pzWYH3GS0XBWvJr7
yLg8/1F1J18l4pk1lXuhM1CaQkJPixN/jvXKlGMpVpa8u7CvSkj9CGshIIV92e7tOvxeBXG
hGFIrN6Sp0ZPa5Jw1gfsdEzBWmbGb4BuE4d3JbdKtszHe1jllZTjsqTBvJtymFCwFpbxpRM
77nAouzE+MnnBAiazK++rYZ9Flw4B4mODgrWkpG5I1nHf1gDFrPa1gveRNmQc+5jnOL2L/p
DqzoGkN2mArpChFgrWXD3eS5J38KDJjDTKsMG4aaDlrXTjr1UdJkJPTLpCChYBAEmzSqcHO
X8utySZXV65AFBFGezjgULBS1dIwaIflDzehVVeVZHFiIN/VFEGoZtVtyUxbtwrpGDNDb3f
heUH26Z4Nq3bkhw5TKT9dtciqihDtynpWN2mK6RgzS/vemH5QemU9kZF0tohX6Er8VteSTm
WPQlOZa5w4gwRQsFaZD/Yu5APLOhdyvs6XOfqu+faVhFlOKsrfwXjRRZHzFOwlumeKbkqr2
xaVUmOdL3IiEPA5ZXmhPn4b2edy1gUrOVh/O2uaY/Vu2TEITi1eiCPMrRNnD9XC9Yz0Zgnc
3SFFKxl9YPd5oT+Su2nkgQjIw7TklhR7ldMbOBzQldIwVpOxu+Z8SWScY7K8iKLEQf3bFTl
UYZWdZjXVT4zTLrCGD16eAlm6QfdCJZ9WEdYLbYjDmG3FU/mRqoJD90EV3+Ga//o5aUPS77
m2QiFrbQm6l24+ok6B+g2R0pj2xWy9SgFa6HV6o74kO9Ykx/vNsdlyficfGVkanRIgpV/4E
uw3v/E4xZBMheYYKn2VZ0HcfS0quK6YaaE4/t8U9MSLlN55X4aRedAXouxVZab54Q0ytBtT
nH933KvkIJFwdIEGsaRVjeZEiMOHsurRmWKyTfdlrj1wb1CCtZy+cHT2nSjorotuWbFvMj6
w6/xhxN81xL/G/zsvY7ks384wfdBDHBURRmkB3EmukIBHpOaBVzDmlF55Wa5ffyeyZZF4Vs
```

```
rILM79e0XGb/5JX7zS8nHt+r92rDz79gvhPPWVkcZpF0S9cgTpHf51maFtQSCpTqOo0d1WC
fPQRUyVFGGs7ouKaq5+IJmJdJYv8PLTMFaDj/ojcZDyd5ZMkd7IqKKMsDHqEcGsihYS+oHT
0zvX016v3FQhYBqrV1/EGeCKxw7pkPBomAtGokV8W3dbXq/Z6A4rMNpYE5Wb8mjDPA9SZuu
cOb3Ey9B6OVVUH5wwFEZW3Xxg5kSTkxfUmjj/MrCdz7+ovpvclxYo2HTVKqVz5xtqyo6zfW
il+VIQsGaGz/4xnevBelhHQD5Cl7eDqA88fCpcX6cns0Fv3JPHmUQWrZ7Y/yYDvcKaQkX2Q
+6P46j5+uS5IN2xCEO9C7xrTWbC36toiyOpgq+KS25SVfICmtpyqsTM5ivbA/7HN8Iy1emj
qQKOGu0lIHrj+SfEhD+5mFJ0t85AlQDJrrNwA6Kt01xuZCukIK1sILlIS+qolGRLJDZEQc/
N6dmxqfmU85dufbTANbpPKCa3wXfa+3Co6JjIWX4coWzWt2jJSRT+EGftc/4nSNdlMmWo86
R5ivDg3XdlryBVwR8ZCrVIdiTACdjrnBaJx7g24CCRcIqrwKvO1pVifNKpCPtoZwyRlrQfD
0jM6iJMgQuoEyQUrAWX7B6F8ELVu8S38jMTqYUXS8BZ4ag8VBnGyP7NgQb6z/qMX7ZhV/le
pGnoyhYMeP/vouRHxzw5rG80V0008CcZrBzEORS0VSoogxQDBz0D6fpULAWSrAi8IPDukYm
E2uF0LfbBTPooQVCIGiiDG0zrEbG7ac8pkPBWiCEwEG3GeLOd/up3IiFXWQ5Xdjx/ZntfKm
iDEC4FR9dIQVrQUhmxQXgsLf5pXem0JE9PDN4/jyAELnnS62JMoTa8P7EpCukYC0EH4QZv5
JiH9YZJ6SIg9MM9i5nZgY1VWQgB3EmXnNh9ZCCRcGaSz4cvYE7VhQjoaSHdUKKODjNYIDzu
KZl9ZZSI76pRJF1oiukYC2CH3TGoBHccRw99mGdcQKPODjN4Omz2YTabVRa3G3izeMovoHx
c+wssihYc+8H30Z1Szcq8tBmgKvv8TGDmV3xweC8DtEwPk2HgkXBmm8/eFoLd+lXuH+kCzc
BRhycZtAqzibUDiCxoiyvzuqRjuQQyuf1Ilu/UrDm2Q9G7Jikh3WCKrKcZvDN41BC7X/+Nz
Bq+Nk3yurJZnx6UPTllap8/oBFFgVrfv1gxILVu5QfnUvmcOWe3y8+CBB0DuRHgvyI1F//C
p9+i7/6Bdbv4E/zuv5/yayyH3QYB3EmVrXCr/jDEu8DCtZ8+sG2OYNz+e2n8m27a76ngQ3+
eYDtrlZv9UXqp3+BRMrVP9FUi1/PQiwEwUoZdIUULPrBaZAeoAtqUEXj4SzbOWmiDG0zuuV
C4bcsyDddIQVrDhCO43iblhrMLfRMmSP1+fCP4ITz//4WHUuZ7dpQJ0VndfR6vHkDXSEFa/
4E68Sc5Tejuns/Mn3dmVY4tUOvg9//J379C/zbTdQ/wN7HcsHSRBla1dmUV3SFFKy5JHVD7
HAS9nEcPefP5YZ0rTDd8BtBBIMKtf/oJwDwP/+N869w/Hf44n3861/iP/4WFy+U/0QTZfB/
EGe9qOyo5bKkFa4MXWE4sKd7OOVVtxnFcRw9x2X5cs+miRdXXX2Fb62RwRMB5hga/4Df/2o
6+dNEGfwfxLle7ddEnqOwp7WRY9gfliJK27PCIh4f0YJDmTmqwzruIw69C5zVh/8FyG//aT
q10nRl8H8QJ1/pq1VmVzKIyCXCpaYrpGDNkx98W4vFN3ZUlucPrlXm7JhueE2vEukRKfS8k
do5EDdPPWsfoWBF6gfP6gEvAKcM5Cv9/zIl5a0rKZEu5bVeUBGHaFi9pbz5/R/E2aiOaHcy
611oTkwKVti89+7dO14Fd49QC3sfyz+183qkwjosBXacba2AfEVcJrdlSHUKR9SmFdxsyjX
uRW6WO2vu+eRL5USc/YKvaHvKwPYriZV+kfPy1ZJZ7Iz63D1DuZT5c953rLBi4gcDyYsmc9
g08cmXkk29xAryD3CzqbyNBXVTzbnyE3GIrnrdVf6YpzW/B3Gc247dVl++PRdZ3Za40qf5O
rM6N07Boh8U7yKfO1a2VO28njCeM7GCT750dWupDuv4iThEQ2JFZ119TsRZL478+F+Xhsth
nv2ysPSu6TbzLYc/U7BmgvCm9Bm/ShnYtiRS1TlA4yEaD3H+fEQQN5+46imq2q3fqMb62mb
Lyvld/g/iOM8k2mcDBl/Tc5ElFNfJXHQDIilYxIVa3Rm5o3wex0kZ2KqL+3ftp3hxFXsGGh
U0Ktgv4Is0Xt4eytaVe5MrAlXT95Qx9Zj1yNBEGXoXk+c5pwydZR5EGWzXPCjWfBZZvUvxi
cWldwrWbHjXm1xe+Vy92jRH1KpzgL2P5U3Tz+ojp2TyD5SVyADV9r+wTRYfNFGGVnWC706k
YdTwyZfYqktkS4gytKrDKzxw9EEVWexBSsGaDb3fTRYsP3lRofl65wD7BV1fBGFH302RJbW
rwt0bEzRRBjcHca79UECt3pLIllOju60RKXd+cW9F1umzkQV1ukIKVoz8oLME8Hkcx6l9vU
vsFyZvJDnv29XC5JdQFVlOfxSf8krFUXlCeZXMiWLnlC3BBY+30BqUb56LrBO6QgpWHAUr0
OV2Z49NVUJdoGMNb103iqNq+o7wx0RPV2yqowzd5uSMW7eJPUOymDiQLWc1NL6057/Icr9X
SChY8ypYmnUQvWYNcBPLUk3WEfb4Z0ggUYZuE1YR1meSWmxgBp1r7SrF8VZkdQ5Glh2Tubj
HRyhYS+cHO5bfXXan9LhPFTrvBDfHiVWHdRCbiIMmynBWn24T9rSGr3LKo9HfXygX9Z11nL
ciS7jIbOlHwYpXeeW/PcP3DpHSz4xRlVQu+x84N8WcxCHikFjR7QB4OOdsByBe3pYsLyaz2
H6FTVOuj4PX8lZkveVeIQUrzoI10cQl0hNaxDkrLDfbdon0yMKT+0Mqvcv4Rhw2qsqqx89B
nLM69gx5CZzZxc5ryev6LLKEGauJdGCjISlYxK8fnHgcZ72Im01dh1+MtsfL7E7OVW1UR/b
LT8wpvn/VYZ3ZRhxSN3S1jM+DOGuF4b6EcFoAwJV7uNkUk1+DqtlbkSUU3SyyKFhzU14Zn/
crF826eO9iZP9r09S1kcmWR+zb6bOpl/xVh3VmGHHQ7FT6b9k+qJJ6l3hVxJ4h7jYOjpQPt
KljDWs6D0UWE6QUrFiQWBl53gpCI7d7Pyyg6B/UDUer39Vb2KpLNCuRxkYV1x+NfHEPjX1V
h3Uwo4jD+h2lmvufiOM85m235ek2cVjCy9uizUysYPMJdn6QLT8rWcI0HbpCCtZ8lFdOd5C
6oSuy7LvIaZGcD/y1AjIlbFsjDY57l97HmqpM1kwiDvryymcDDLuNcrclbpKe1bFfwOFd8e
sns9h80k9s+SmyGMgKGjbwc81ZvT+Rwfh85J3npodcIo2bzb4rPH+O/cIEQRQOFWqe4frjO
xPZfCIvHAY/bDTkHyjlwE6BBjVAO5nTLd7lH8i+gdbQIx/endp6f3o+LJN7F/hitf//mq6E
hBVWkH7QqVbdpqutK2d4WjO7eFCyfZVD4+GEgz7+1QrqoMBaIbqIw8QoQ1BqBXXyw3adL65
KfpvOFT2fK1l0hRSsOfCD475m05zwdLXvnz0DL66i8VByx3YOsGcEMDJeOPo7UvVENahCE2
VwcxAnQLpN7Bfw8rZygd/DShb3CilYMRKsN67Xp3sXw/Upu1mopn2KfXzXqGHnNfIPROGwT
WVQM01VveGTuSgiDvoog+cpgT69/4scju8HU9kJx3TWi3M2ryhmcA1rmvexVcSnjntbM5ZC
```

```
xaY5YrXsjaSOhY6FRBopA8kcUoauIUnjod8tM0kxpVhC6l0o85ZBoVnKiXgdTeJV09iojvy
+vM2nEC6vPaOEa1gUrNAFq22OpNWPyl5GeAqa5Z7z52hUAh5oOkAY/DOgbeLwbmjl6h0Yak
/tcyJOYDWggY1qf9vUw6I7xqbpnNZgfUbBoiWM3A96a89wWJrabpw+w8vb2C+EpVZQr75nS
iFGHDRRhrYZC7Wy6+j9AqzPvKRzB3WZc7WRrpAVVhRc/AvSPxOfk37sxnoRawUkc0ikJR6w
28J5HWd1nNYiGgm1/Up+cigka3blnq4/xLzMTPT2wx6WkCmxwqJghcnvj/DTDXElItgVk/c
NAPjWms3QOjtbr6oKA/5h1eNdAbSqOL6/UG+exMrI6udpDYk0BYuCFSZ//B3+5M/6/9+7wF
e5IPNBMUG1sBJsehPA9Ue6iTgLeW2FvHHHcttEiDjgGpZrBmqFIKalxhPVYZ1gIw6a+V0I4
iBOPBEie1QrCtbM3nwLQ+dAua6cLQfWxeEjU/mpbhONh4t5bdtPOZ6egjULuk1f01Jjjqrp
eyLtfYC7k9VburWbwCNmfM5RsFheLbQcqyfrCJMTvaFpu9qxIj2IEz0nJu8eClb0tf2iv+1
Uh3Xgu1XWlXu6TqpH5QW/sOfPAztQRcEiruhYvqalzgW9S3yjsGZrBe/9BhIruKZ2fGf1uC
RFWZ5TsFjVzxlvHitrAc9FluawN3y3bGd5TsEiEt4uzRNStf6dzMkb3enRRxna5uLXrf0K/
SCApkAULOK2nl+k8yITaoGnyqOL2fLUp+E+Mr2II4t0QsHyJVhLhUpH7L4r7pkYZViex8BS
FekULApWpGgm60wVcdCom7N59JLQbXHp3TMJXgK3vOvBqKF3gY6FbhPdJr5rLn5p8HVppJe
Tk+tVV10c9ONjF/UgzshNtoKUgR+nkTKGbRqJJ3j42f8Ds4luEx2rr2XfX6BjLdRNqJqsA8
AqTgj967sydJt4cXWh3gypG8M2DKsFAGzJQMGaE2wzdV7v/3/vYl43wpJZbFty0ZmoOJr5X
Qiha02U1+QnOSRz/ZbWdmsgTWiDULDmkt5Fv93VfPlKje40KsrjykJr4HFBn23Lds9ujoaO
gkVfGWtfqXF2mvZVQgcogZi0bKebo2CRBfSVmo7G0gahmv6lsy2v6OYoWMuL7ewiftPPyle
qJutA1oJd1SFe9fcXz83ZD5vvmlPPXiUUrBBpm8Pooz1gZmAr7LtlYXylZiqXUDFldnVtZA
IfHTZbN6e67IkVZMvIllm+UbDiR6uKRkWuDs5HfTI39CPz6Cs10/QGa1L6KIOf4ayzdXNTF
baZXWxUKVUUrBhjh7bdJyHt289pW+LvKzUrU4OIgz7KoNlVjJub8ybxmV3kK9xJpGDNj2wd
lX3Fi2LuKzV7f0dlvK3pogzjW4rxdHOef3H5CvcWKVhzSLeJ43KQrd/j4yuTOeUqsl21ae7
YjoXT2tyUk1N51Y9MShUFa845q6NRCTdtNFtfGc9rjgiDIMks8hXuA1KwFojTGo7LUcfZZ+
srI3Nz3/3g6aKP2nITkIK1yLRNHJVnHF6fua/06eZsVYrDYaYr93CtQqmiYC00024jRkZMf
KUtSQM3B8RxLAU3ASlYSydb31Tw5vEcfKsh+cqZuznPV2OjyhHzFKylpNtEozKXzVXc+8p4
ujkPpG7gepWbgBSspSeCbcRoGA+LzkX3GDdmmZuAsXpc8hLMkrUC1uo4q+Pr0nINYpiLQjJ
b1kX2ySzgEIp4yNZOE5tPkMzyYsSlYLzZpFpRsIiaTAnbFvIPph75R4L8Lexi5/WEIdWEgk
UAIJFGvoKbTS+jlYlPVm9h5zU2TUYWKFhketnaeY3MLi9GRFL1yZfYqlOqKFjEK8kcNk1sv
+qHoUgoFzmLzSfYqjOyQMEiQZAysFXHJ19OMWaZuCpjV3D9EXbYv5iCRQJnrYBti9uIgUmV
vYzBIcUAAAIqSURBVAmYLfNiULBIaGRK2GlyG9HfNdzFtsVNQAoWiYrBNiJlayq4CUjBIjM
yNWnkK9i2uI3oVqq4CUjBIjPG3kbcec1tRPUlysL4nJuAFCwSJ9mytxEpWyNF6Ao2n2CnqZ
yXQShYZGasFbBV5zZiX6rsTUDmFShYJNbY24jXHy3venxmt39omZuAFCwyH2TLy7iNuH6nv
wlIqaJgkXmzRcu0jWhvAho1bgJSsMg8M9hGXL+zoD9gtp9X4CYgBYssjmwZtUXbRrQPLe80
KVUULLKI2NuIxudzv41obwJuW9wEpGCRRWe92O/FPKfr8VfucROQgkWWjExp/rYR7c7FG1V
KFQWLLB+DXszx30a0NwF5aJlQsChb/W3EeMpW6gY3AQkFi4xipx9itY1obwJuW5QqIj5keQ
kIEJuRrhxfSlhhkSlka4YjXTm+lFCwyNREP9KV40sJBYv4sGY/bCNeuRfuC63ewvYrbgISC
hYJQrY2qmFtIw46F6cMXmlCwSIBEfhIV44vJRQsEi6BjHTl+FJCwSLR4XmkK8eXEgoWmQ3T
jnTl+FJCwSIzZjDSVQPHl5JAee/du3e8CsQX3Sa6Y730pB8khIJFCKElJIQQChYhhFCwCCE
ULEIIoWARQggFixBCwSKEEAoWIYRQsAghFCxCCKFgEUIIBYsQQsEihBAKFiGEULAIIRQsQg
ihYBFCCAWLEELBIoQQChYhhILFS0AIoWARQkjA/D87uqZQTj7xTgAAAABJRU5ErkJggg=="
;
```

Insert an image

1. Open the file **./src/taskpane/taskpane.html**.
2. Locate the `<button>` element for the `replace-text` button, and add the following markup after that line.

HTML

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

3. Open the file **./src/taskpane/taskpane.js**.

4. Locate the `Office.onReady` function call near the top of the file and add the following code immediately before that line. This code imports the variable that you defined previously in the file **./base64Image.js**.

JavaScript

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

5. Within the `Office.onReady` function call, locate the line that assigns a click handler to the `replace-text` button, and add the following code after that line.

JavaScript

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

6. Add the following function to the end of the file.

JavaScript

```
async function insertImage() {
    await Word.run(async (context) => {

        // TODO1: Queue commands to insert an image.

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

7. Within the `insertImage()` function, replace `TODO1` with the following code. Note that this line inserts the Base64-encoded image at the end of the document. (The `Paragraph` object also has an `insertInlinePictureFromBase64` method and other `insert*` methods. See the following "Insert HTML" section for an example.)

JavaScript

```
context.document.body.insertInlinePictureFromBase64(base64Image,
Word.InsertLocation.end);
```

Insert HTML

1. Open the file **./src/taskpane/taskpane.html**.

2. Locate the `<button>` element for the `insert-image` button, and add the following markup after that line.

   HTML

   ```html
   <button class="ms-Button" id="insert-html">Insert HTML</button><br/>
   <br/>
   ```

3. Open the file **./src/taskpane/taskpane.js**.

4. Within the `Office.onReady` function call, locate the line that assigns a click handler to the `insert-image` button, and add the following code after that line.

   JavaScript

   ```javascript
   document.getElementById("insert-html").onclick = () =>
   tryCatch(insertHTML);
   ```

5. Add the following function to the end of the file.

   JavaScript

   ```javascript
   async function insertHTML() {
       await Word.run(async (context) => {

           // TODO1: Queue commands to insert a string of HTML.

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

6. Within the `insertHTML()` function, replace `TODO1` with the following code. Note:

   - The first line adds a blank paragraph to the end of the document.

   - The second line inserts a string of HTML at the end of the paragraph; specifically two paragraphs, one formatted with the Verdana font, the other with the default styling of the Word document. (As you saw in the `insertImage` method earlier, the `context.document.body` object also has the `insert*` methods.)

JavaScript

```javascript
const blankParagraph =
context.document.body.paragraphs.getLast().insertParagraph("",
Word.InsertLocation.after);
blankParagraph.insertHtml('<p style="font-family: verdana;">Inserted
HTML.</p><p>Another paragraph</p>', Word.InsertLocation.end);
```

Insert a table

1. Open the file **./src/taskpane/taskpane.html**.

2. Locate the `<button>` element for the `insert-html` button, and add the following markup after that line.

   HTML

   ```html
   <button class="ms-Button" id="insert-table">Insert Table</button><br/>
   <br/>
   ```

3. Open the file **./src/taskpane/taskpane.js**.

4. Within the `Office.onReady` function call, locate the line that assigns a click handler to the `insert-html` button, and add the following code after that line.

   JavaScript

   ```javascript
   document.getElementById("insert-table").onclick = () =>
   tryCatch(insertTable);
   ```

5. Add the following function to the end of the file.

   JavaScript

   ```javascript
   async function insertTable() {
       await Word.run(async (context) => {

           // TODO1: Queue commands to get a reference to the paragraph
           //        that will precede the table.

           // TODO2: Queue commands to create a table and populate it with
   data.

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

6. Within the `insertTable()` function, replace `TODO1` with the following code. Note that this line uses the `ParagraphCollection.getFirst` method to get a reference to the first paragraph and then uses the `Paragraph.getNext` method to get a reference to the second paragraph.

   JavaScript

   ```javascript
   const secondParagraph =
   context.document.body.paragraphs.getFirst().getNext();
   ```

7. Within the `insertTable()` function, replace `TODO2` with the following code. Note:

   - The first two parameters of the `insertTable` method specify the number of rows and columns.
   - The third parameter specifies where to insert the table, in this case after the paragraph.
   - The fourth parameter is a two-dimensional array that sets the values of the table cells.
   - The table will have plain default styling, but the `insertTable` method returns a `Table` object with many members, some of which are used to style the table.

   JavaScript

   ```javascript
   const tableData = [
           ["Name", "ID", "Birth City"],
           ["Bob", "434", "Chicago"],
           ["Sue", "719", "Havana"],
       ];
   secondParagraph.insertTable(3, 3, Word.InsertLocation.after,
   tableData);
   ```

8. Save all your changes to the project.

Test the add-in

1. If the local web server is already running and your add-in is already loaded in Word, proceed to step 2. Otherwise, start the local web server and sideload your add-in.

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

  command line

  ```
  npm start
  ```

- To test your add-in in Word on the web, 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 Word 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!jkcH7spkM4EGgcZUgqthk4IK3NOypVw?e=Z6G1qp`
  - `npm run start -- web --document https://contoso-my.sharepoint-df.com/:t:/p/user/EQda453DNTpFnl1bFPhOVR0BwlrzetbXvnaRYii2lDr_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.

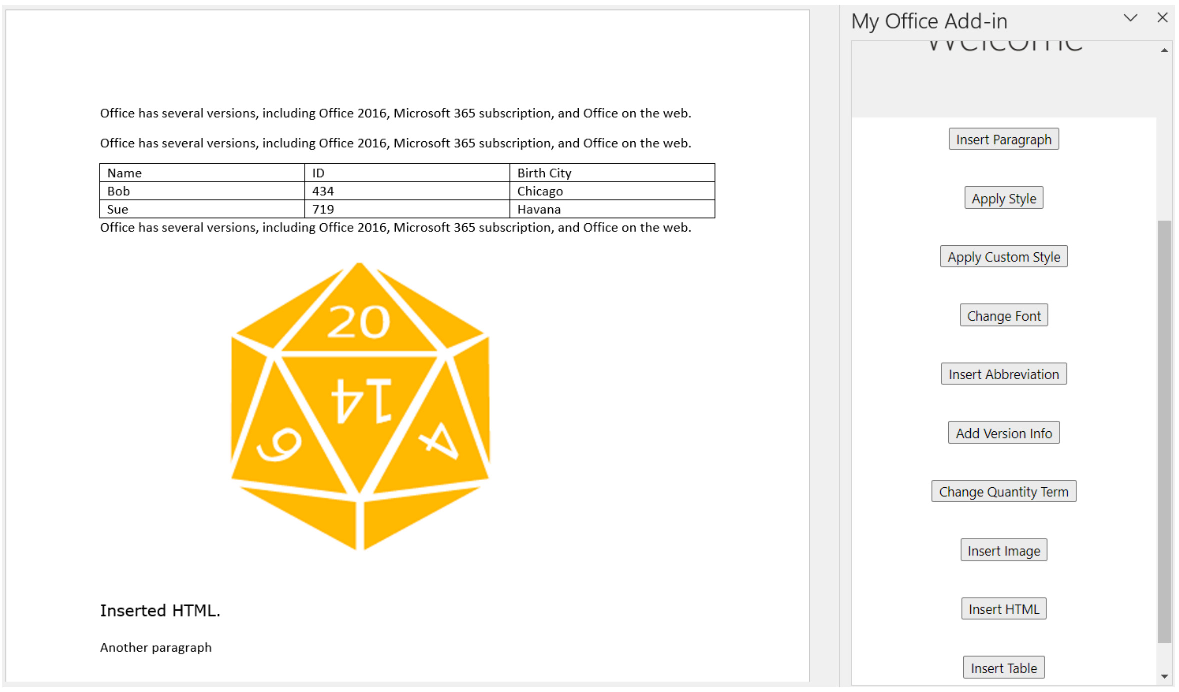
2. If the add-in task pane isn't already open in Word, go to the **Home** tab and choose the **Show Taskpane** button on the ribbon to open it.

3. In the task pane, choose the **Insert Paragraph** button at least three times to ensure that there are a few paragraphs in the document.

4. Choose the **Insert Image** button and note that an image is inserted at the end of the document.

5. Choose the **Insert HTML** button and note that two paragraphs are inserted at the end of the document, and that the first one has the Verdana font.

6. Choose the **Insert Table** button and note that a table is inserted after the second paragraph.



Office has several versions, including Office 2016, Microsoft 365 subscription, and Office on the web.

Office has several versions, including Office 2016, Microsoft 365 subscription, and Office on the web.

Name	ID	Birth City
Bob	434	Chicago
Sue	719	Havana

Office has several versions, including Office 2016, Microsoft 365 subscription, and Office on the web.

Inserted HTML.

Another paragraph

Create and update content controls

In this step of the tutorial, you'll learn how to create Rich Text content controls in the document, and then how to insert and replace content in the controls.

ⓘ Note

Before you start this step of the tutorial, we recommend that you create and manipulate Rich Text content controls through the Word UI, so you can be familiar with the controls and their properties. For details, see Create forms that users complete or print in Word.

Create a content control

1. Open the file **./src/taskpane/taskpane.html**.

2. Locate the `<button>` element for the `insert-table` button, and add the following markup after that line.

   HTML

   ```html
   <button class="ms-Button" id="create-content-control">Create Content
   Control</button><br/><br/>
   ```

3. Open the file **./src/taskpane/taskpane.js**.

4. Within the `Office.onReady` function call, locate the line that assigns a click handler to the `insert-table` button, and add the following code after that line.

   JavaScript

   ```javascript
   document.getElementById("create-content-control").onclick = () =>
   tryCatch(createContentControl);
   ```

5. Add the following function to the end of the file.

   JavaScript

   ```javascript
   async function createContentControl() {
       await Word.run(async (context) => {

           // TODO1: Queue commands to create a content control.

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

6. Within the `createContentControl()` function, replace `TODO1` with the following code. Note:

   - This code is intended to wrap the phrase "Microsoft 365" in a content control. It makes a simplifying assumption that the string is present and the user has selected it.
   - The `ContentControl.title` property specifies the visible title of the content control.
   - The `ContentControl.tag` property specifies an tag that can be used to get a reference to a content control using the `ContentControlCollection.getByTag` method, which you'll use in a later function.

- The `ContentControl.appearance` property specifies the visual look of the control. Using the value "Tags" means that the control will be wrapped in opening and closing tags, and the opening tag will have the content control's title. Other possible values are "BoundingBox" and "None".
- The `ContentControl.color` property specifies the color of the tags or the border of the bounding box.

JavaScript

```javascript
const serviceNameRange = context.document.getSelection();
const serviceNameContentControl =
serviceNameRange.insertContentControl();
serviceNameContentControl.title = "Service Name";
serviceNameContentControl.tag = "serviceName";
serviceNameContentControl.appearance = "Tags";
serviceNameContentControl.color = "blue";
```

Replace the content of the content control

1. Open the file **./src/taskpane/taskpane.html**.
2. Locate the `<button>` element for the `create-content-control` button, and add the following markup after that line.

   HTML

   ```html
   <button class="ms-Button" id="replace-content-in-control">Rename
   Service</button><br/><br/>
   ```

3. Open the file **./src/taskpane/taskpane.js**.
4. Within the `Office.onReady` function call, locate the line that assigns a click handler to the `create-content-control` button, and add the following code after that line.

   JavaScript

   ```javascript
   document.getElementById("replace-content-in-control").onclick = () =>
   tryCatch(replaceContentInControl);
   ```

5. Add the following function to the end of the file.

   JavaScript

```
async function replaceContentInControl() {
    await Word.run(async (context) => {

        // TODO1: Queue commands to replace the text in the Service
Name
        //        content control.

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

6. Within the `replaceContentInControl()` function, replace `TODO1` with the following code. Note:

   - The `ContentControlCollection.getByTag` method returns a `ContentControlCollection` of all content controls of the specified tag. We use `getFirst` to get a reference to the desired control.

   JavaScript

   ```
   const serviceNameContentControl =
   context.document.contentControls.getByTag("serviceName").getFirst();
   serviceNameContentControl.insertText("Fabrikam Online Productivity
   Suite", Word.InsertLocation.replace);
   ```

7. Save all your changes to the project.

Test the add-in

1. If the local web server is already running and your add-in is already loaded in Word, proceed to step 2. Otherwise, start the local web server and sideload your add-in.

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

     command line

     ```
     npm start
     ```

   - To test your add-in in Word on the web, 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 Word 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!jkcH7spkM4EGgcZUgqthk4IK3NOypVw?e=Z6G1qp`
- `npm run start -- web --document https://contoso-my.sharepoint-df.com/:t:/p/user/EQda453DNTpFnl1bFPhOVR0BwlrzetbXvnaRYii2lDr_oQ?e=RSccmNP`

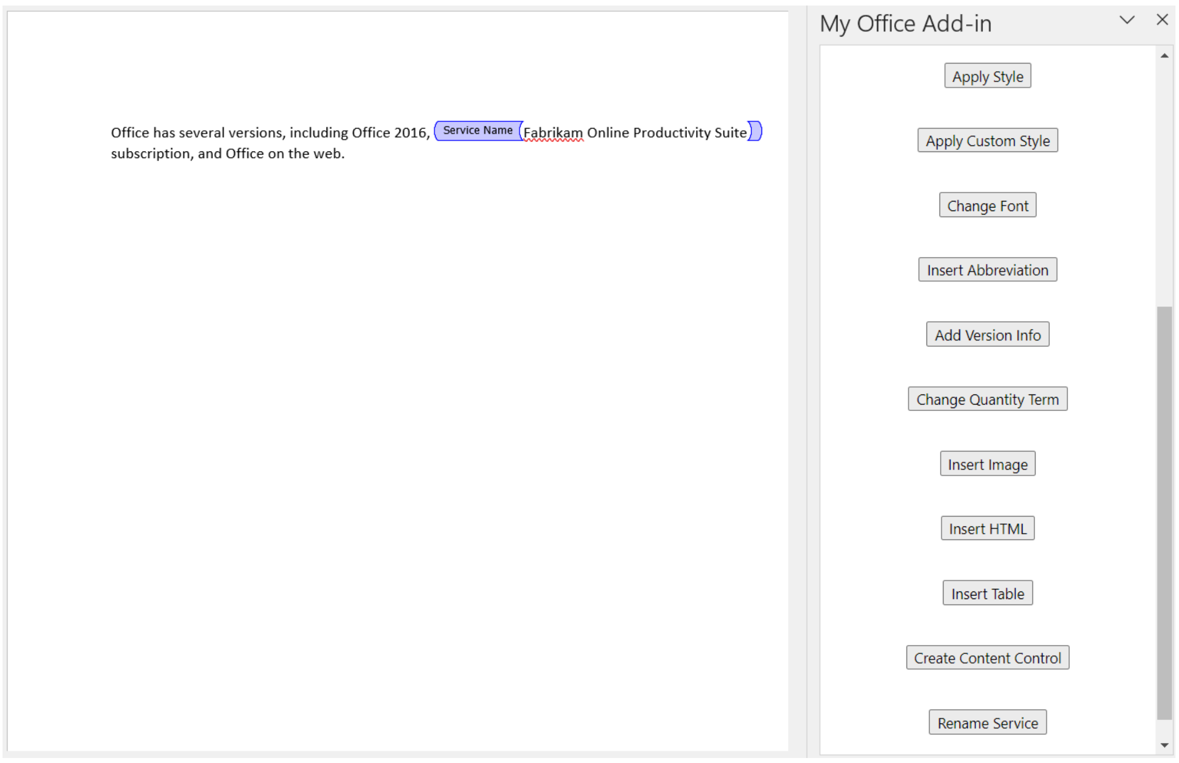
If your add-in doesn't sideload in the document, manually sideload it by following the instructions in Manually sideload add-ins to Office on the web.

2. If the add-in task pane isn't already open in Word, go to the **Home** tab and choose the **Show Taskpane** button on the ribbon to open it.

3. In the task pane, choose the **Insert Paragraph** button to ensure that there's a paragraph with "Microsoft 365" at the top of the document.

4. In the document, select the text "Microsoft 365" and then choose the **Create Content Control** button. Note that the phrase is wrapped in tags labelled "Service Name".

5. Choose the **Rename Service** button and note that the text of the content control changes to "Fabrikam Online Productivity Suite".


Office has several versions, including Office 2016, Service Name Fabrikam Online Productivity Suite subscription, and Office on the web.

Next steps

In this tutorial, you've created a Word task pane add-in that inserts and replaces text, images, and other content in a Word document. To learn more about building Word add-ins, continue to the following article.

Word add-ins overview

Code samples

- Completed Word add-in tutorial : The result of completing this tutorial.

See also

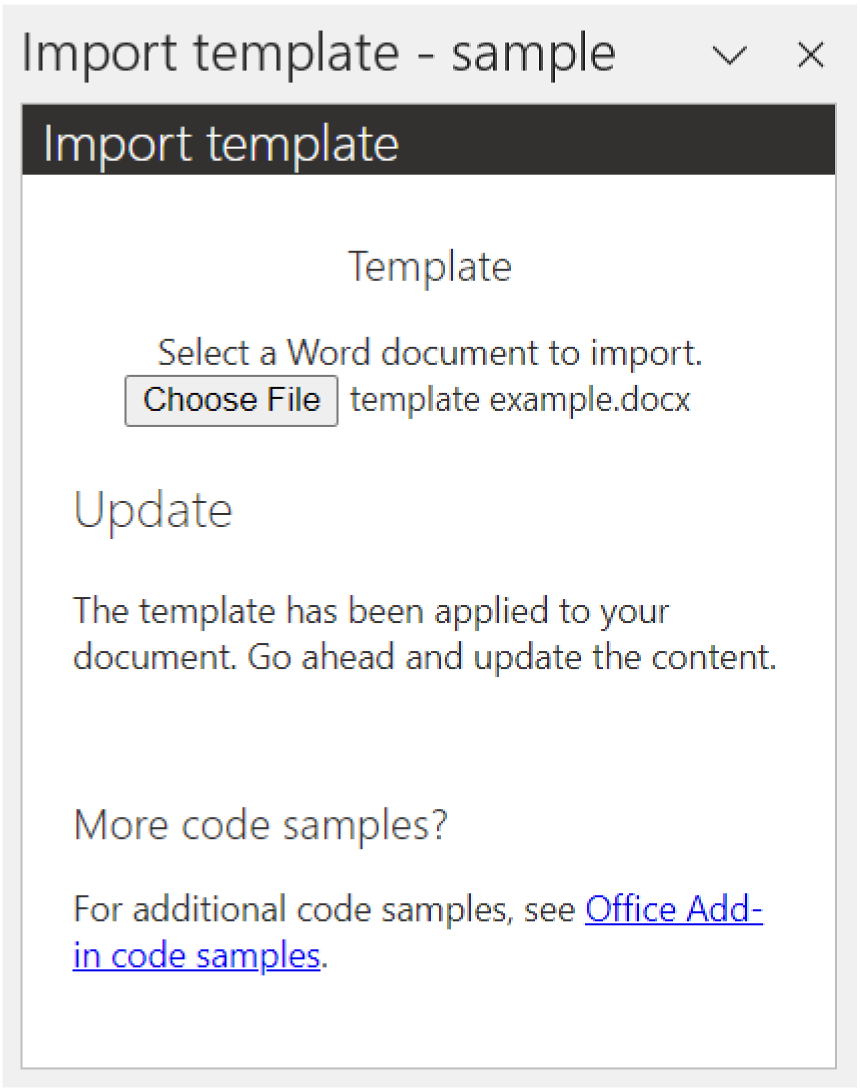
- Office Add-ins platform overview
- Develop Office Add-ins

Sample: Import a Word document template with a Word add-in

Article • 03/11/2024

Templates enable users to quickly create consistent documents for their organizations. Templates can include company information and other critical details that users need for compliance, legal, or other reasons.

This article features a sample add-in that imports a .docx file to use as a template in a Word document. The add-in replaces the current document's content with the content from the template.


Import template - sample

Import template

Template

Select a Word document to import.

Update

The template has been applied to your document. Go ahead and update the content.

More code samples?

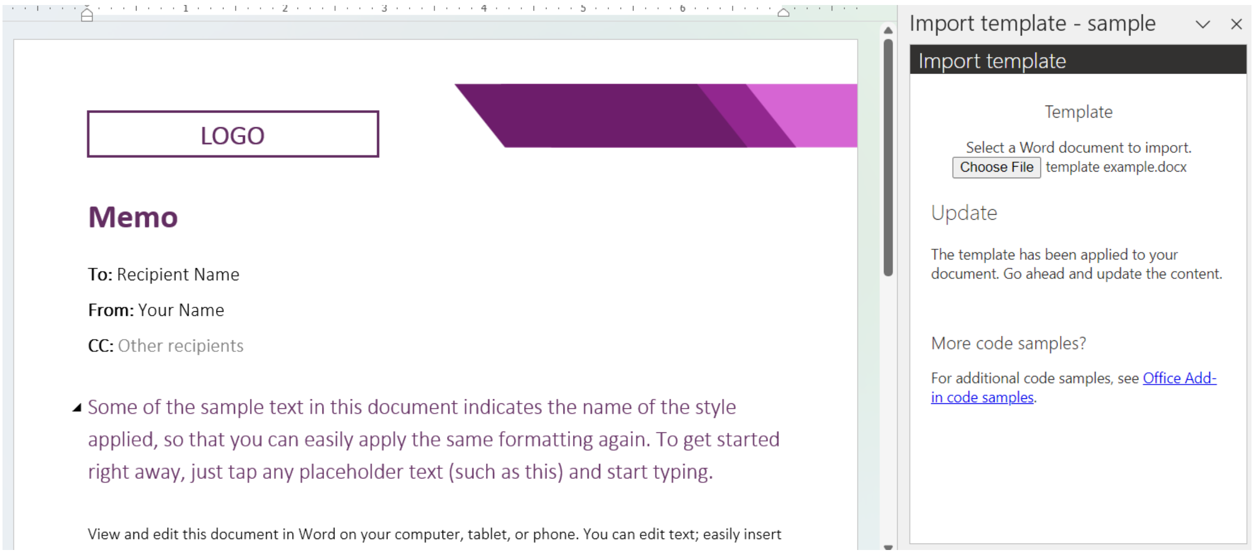
For additional code samples, see Office Add-in code samples.

Prerequisites

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

Run the sample code

The sample code for this article is named Import templates in a Word document. To run the sample, follow the instructions in the readme.



Memo

To: Recipient Name

From: Your Name

CC: Other recipients

Some of the sample text in this document indicates the name of the style applied, so that you can easily apply the same formatting again. To get started right away, just tap any placeholder text (such as this) and start typing.

View and edit this document in Word on your computer, tablet, or phone. You can edit text; easily insert

Template

Select a Word document to import.

template example.docx

Update

The template has been applied to your document. Go ahead and update the content.

More code samples?

For additional code samples, see Office Add-in code samples.

Key steps in the sample

1. The user chooses a .docx file they'd like to use as a template.
2. The add-in reads the template .docx file then uses Document.insertFileFromBase64 to replace the current document's content with the content from the template file.
3. The user can make updates to the content of the current document.

Make it yours

The following are a few suggestions for how you could tailor this sample to your scenario.

Manage user settings

Enable single sign-on (SSO) in an Office Add-in to support persisting user data and settings across multiple documents. If your service provides or hosts a document template library, an authorized user can access and apply a template in their document.

You can also persist add-in state and settings in the user's current document.

Ⓧ Caution

Don't store sensitive information such as authentication tokens or connection strings. Properties in the document aren't encrypted or protected.

Provide templates

Provide personalized or company-approved templates for users. These templates can be made accessible from a shared location as part of an authenticated experience.

You can use content controls, fields, and other components as building blocks in your templates.

Personalize templates

Allow users to personalize or refine templates. For templates that may be useful to others (on their team, in their company, etc.), users can upload to a shared location.

See also

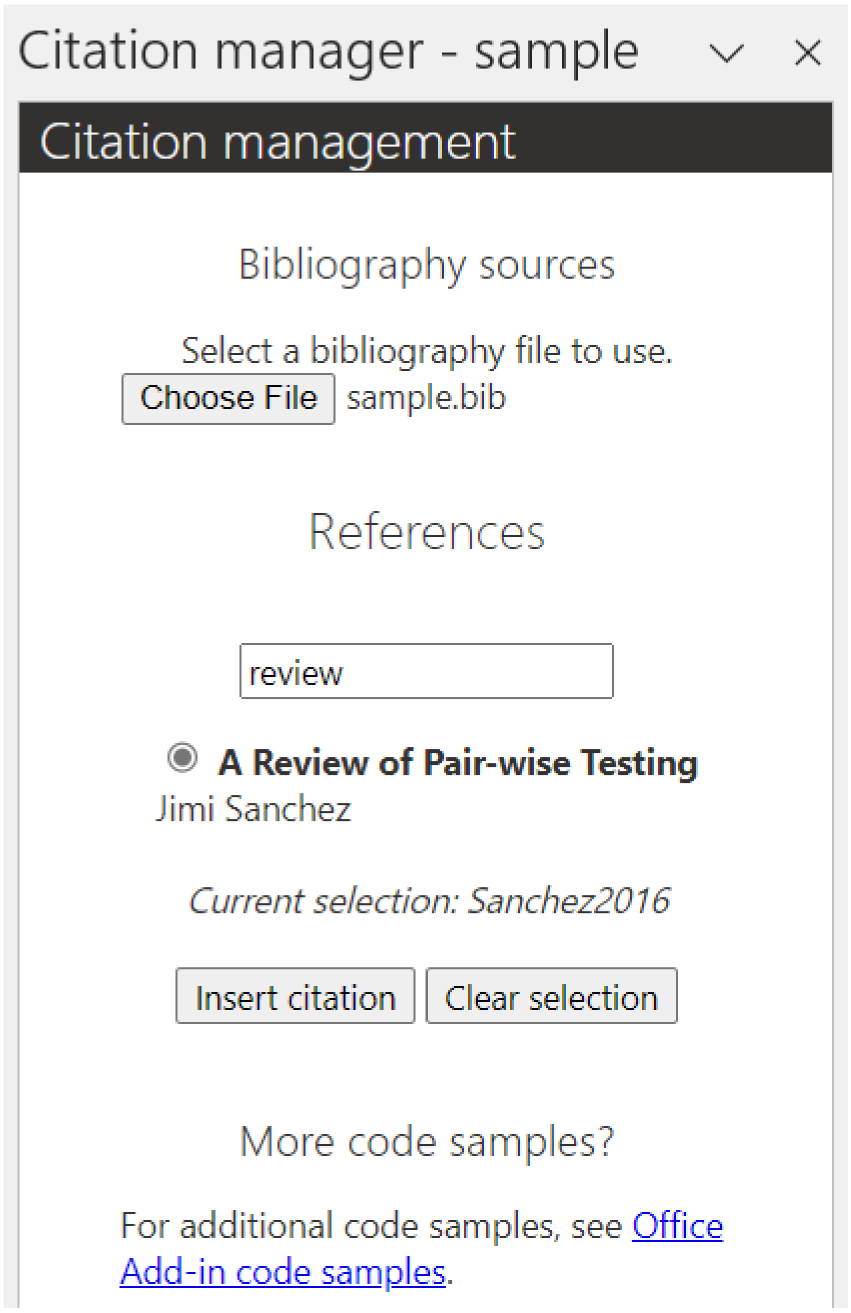
- Office Add-in code samples

Sample: Manage citations in a Word document using your Word add-in

Article • 11/29/2023

Citation management is an important aspect of documents, particularly in academia and education. Each citation style has its own guidelines for how citations should be marked in a document as well as where and how the sources should be noted. Such styles include APA and MLA.

This article features a sample add-in that manages citations in a Word document. The add-in displays the references loaded from a .bib file that the user selects to cite in their document.


Citation manager - sample
Citation management
Bibliography sources
Select a bibliography file to use.
Choose File
sample.bib
References
review
A Review of Pair-wise Testing
Jimi Sanchez
Current selection: Sanchez2016
Insert citation
Clear selection
More code samples?
For additional code samples, see Office Add-in code samples.

Prerequisites

- Visual Studio Code.
- Office connected to a Microsoft 365 subscription (including Office on the web).
- Node.js version 16 or greater.
- npm version 8 or greater.

Run the sample code

The sample code for this article is named Manage citations in a Word document. To run the sample, follow the instructions in the readme.

Key steps in the sample

1. The user chooses a local .bib file that contains the references they'd like to cite.
2. The add-in reads the .bib file then displays the bibliography references in the task pane. The sample uses @orcid/bibtexParseJs to parse the .bib file.
3. The user chooses the appropriate reference then inserts it at the cursor's location (or at the end of selected text) in the document.
4. The add-in adds a reference mark at that location in the document and adds the reference to an endnote. All endnotes are automatically listed at the end of the document.

Make it yours

The following are a few suggestions for how you could tailor this sample to your scenario.

Manage user settings

Enable single sign-on (SSO) in an Office Add-in to support persisting user data and settings across multiple documents. If your service provides or hosts the bibliography library, an authorized user can access and select from that bibliography in their document.

You can also persist add-in state and settings in the user's current document.

⊗ **Caution**

Don't store sensitive information such as authentication tokens or connection strings. Properties in the document aren't encrypted or protected.

Use footnotes

List the references in footnotes at the end of the page instead of endnotes, according to the citation style.

Alternatively, allow the user to choose where they'd like the references to be displayed. If so, you can update the add-in to persist the user's preference using a document property or as part of their authenticated experience.

Update citation style

Update the citation style used to display the references in the endnotes (or footnotes).

Alternatively, provide various style options then allow the user to choose. If so, you can update the add-in to persist the user's preference using a document property or as part of their authenticated experience.

Replace bibtexParseJs

Replace the .bib file parser @orcid/bibtexParseJs with your own or another available parser, especially if this option doesn't provide the functionality you need for your solution.

See also

- Office Add-in code samples
- npm
- @orcid/bibtexParseJs

Add headers when a document opens

07/14/2025

The following sections walk you through how to develop a Word add-in that automatically changes the document header when a new or existing document opens. While this specific sample is for Word, the manifest configuration is the same for Excel and PowerPoint. For an overview of this style of event-based activation pattern, see Activate add-ins with events.

> **Important**
>
> This sample requires you to have a Microsoft 365 subscription with the supported version of Word.

Create a new add-in

Create a new add-in by following the Word add-in quick start. This will give you a working Office Add-in to which you can add the event-based activation code.

> **Note**
>
> For a completed version of the sample described in this walkthrough, see the Automatically add labels with an add-in when a Word document opens sample in our samples GitHub repo.

Configure the manifest

To enable an event-based add-in, you must configure the following elements in the `VersionOverridesV1_0` node of the manifest.

- In the Runtimes element, make a new Override element for Runtime. Override the "javascript" type and reference the JavaScript file containing the function you want to trigger with the event.
- In the DesktopFormFactor element, add a FunctionFile element for the JavaScript file with the event handler.
- In the ExtensionPoint element, set the `xsi:type` to `LaunchEvent`. This enables the event-based activation feature in your add-in.
- In the LaunchEvent element, set the `Type` to `OnDocumentOpened` and specify the JavaScript function name of the event handler in the `FunctionName` attribute.

Use the following sample manifest code to update your project.

1. In your code editor, open the quick start project you created.

2. Open the **manifest.xml** file located at the root of your project.

3. Select the entire **<VersionOverrides>** node (including the open and close tags) and replace it with the following XML.

XML

```xml
  <VersionOverrides
xmlns="http://schemas.microsoft.com/office/taskpaneappversionoverrides"
xsi:type="VersionOverridesV1_0">
    <Hosts>
      <Host xsi:type="Document">
        <Runtimes>
          <Runtime resid="Taskpane.Url" lifetime="long" />
          <Runtime resid="WebViewRuntime.Url">
            <Override type="javascript" resid="JsRuntimeWord.Url"/>
          </Runtime>
        </Runtimes>
        <DesktopFormFactor>
          <GetStarted>
            <Title resid="GetStarted.Title"/>
            <Description resid="GetStarted.Description"/>
            <LearnMoreUrl resid="GetStarted.LearnMoreUrl"/>
          </GetStarted>
          <FunctionFile resid="Commands.Url"/>
          <ExtensionPoint xsi:type="LaunchEvent">
            <LaunchEvents>
              <LaunchEvent Type="OnDocumentOpened"
FunctionName="changeHeader"></LaunchEvent>
            </LaunchEvents>
            <SourceLocation resid="WebViewRuntime.Url"/>
          </ExtensionPoint>
          <ExtensionPoint xsi:type="PrimaryCommandSurface">
            <OfficeTab id="TabHome">
              <Group id="CommandsGroup">
                <Label resid="CommandsGroup.Label"/>
                <Icon>
                  <bt:Image size="16" resid="Icon.16x16"/>
                  <bt:Image size="32" resid="Icon.32x32"/>
                  <bt:Image size="80" resid="Icon.80x80"/>
                </Icon>
                <Control xsi:type="Button" id="TaskpaneButton">
                  <Label resid="TaskpaneButton.Label"/>
                  <Supertip>
                    <Title resid="TaskpaneButton.Label"/>
                    <Description resid="TaskpaneButton.Tooltip"/>
                  </Supertip>
                  <Icon>
                    <bt:Image size="16" resid="Icon.16x16"/>
                    <bt:Image size="32" resid="Icon.32x32"/>
                    <bt:Image size="80" resid="Icon.80x80"/>
                  </Icon>
                  <Action xsi:type="ShowTaskpane">
                    <TaskpaneId>ButtonId1</TaskpaneId>
                    <SourceLocation resid="Taskpane.Url"/>
                  </Action>
                </Control>
              </Group>
            </OfficeTab>
          </ExtensionPoint>
        </DesktopFormFactor>
      </Host>
    </Hosts>
    <Resources>
      <bt:Images>
        <bt:Image id="Icon.16x16"
DefaultValue="https://localhost:3000/assets/icon-16.png"/>
        <bt:Image id="Icon.32x32"
DefaultValue="https://localhost:3000/assets/icon-32.png"/>
        <bt:Image id="Icon.80x80"
DefaultValue="https://localhost:3000/assets/icon-80.png"/>
      </bt:Images>
      <bt:Urls>
        <bt:Url id="GetStarted.LearnMoreUrl"
DefaultValue="https://go.microsoft.com/fwlink/?LinkId=276812"/>
        <bt:Url id="Commands.Url"
DefaultValue="https://localhost:3000/commands.html"/>
        <bt:Url id="Taskpane.Url"
DefaultValue="https://localhost:3000/taskpane.html"/>
        <bt:Url id="WebViewRuntime.Url"
DefaultValue="https://localhost:3000/commands.html"/>
        <bt:Url id="JsRuntimeWord.Url"
DefaultValue="https://localhost:3000/commands.js"/>
      </bt:Urls>
      <bt:ShortStrings>
        <bt:String id="GetStarted.Title" DefaultValue="Get started with your
sample add-in!"/>
        <bt:String id="CommandsGroup.Label" DefaultValue="Event-based add-in
activation"/>
        <bt:String id="TaskpaneButton.Label" DefaultValue="My add-in"/>
      </bt:ShortStrings>
      <bt:LongStrings>
        <bt:String id="GetStarted.Description" DefaultValue="Your sample add-
in loaded successfully. Go to the HOME tab and click the 'Show Task Pane'
button to get started."/>
        <bt:String id="TaskpaneButton.Tooltip" DefaultValue="Click to show
the task pane"/>
      </bt:LongStrings>
    </Resources>
  </VersionOverrides>
```

4. Save your changes.

Implement the event handler

To enable your add-in to act when the `OnDocumentOpened` event occurs, you must implement a JavaScript event handler. In this section, you'll create the `changeHeader` function, which adds a "Public" header to new documents or a "Highly Confidential" header to existing documents that already have content.

1. In the **./src/commands** folder, open the file named **commands.js**.

2. Replace the entire contents of **commands.js** with the following JavaScript code.

JavaScript

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

  Office.onReady(() => {
    // If needed, Office.js is ready to be called.
  });

  async function changeHeader(event) {
    Word.run(async (context) => {
      const body = context.document.body;
      body.load("text");
      await context.sync();

      if (body.text.length === 0) {
      // For new or empty documents, make a "Public" header.
        const header =
context.document.sections.getFirst().getHeader(Word.HeaderFooterType.primary)
;
        const firstPageHeader =
context.document.sections.getFirst().getHeader(Word.HeaderFooterType.firstPag
e);
        header.clear();
        firstPageHeader.clear();

        header.insertParagraph("Public - The data is for the public and
shareable externally", "Start");
        firstPageHeader.insertParagraph("Public - The data is for the public
and shareable externally", "Start");
        header.font.color = "#07641d";
        firstPageHeader.font.color = "#07641d";
        await context.sync();
      } else {
        // For existing documents, make a "Highly Confidential" header.
        const header =
context.document.sections.getFirst().getHeader(Word.HeaderFooterType.primary)
;
        const firstPageHeader =
context.document.sections.getFirst().getHeader(Word.HeaderFooterType.firstPag
e);
        header.clear();
        firstPageHeader.clear();
        header.insertParagraph("Highly Confidential - The data must be secret
or in some way highly critical", "Start");
        firstPageHeader.insertParagraph("Highly Confidential - The data must
be secret or in some way highly critical", "Start");
        header.font.color = "#f8334d";
        firstPageHeader.font.color = "#f8334d";
        await context.sync();
      }
    });

    // Calling event.completed is required. event.completed lets the platform
know that processing has completed.
    event.completed();
  }

  async function paragraphChanged() {
    await Word.run(async (context) => {
      const results = context.document.body.search("110");
      results.load("length");
      await context.sync();
      if (results.items.length === 0) {
        const header =
context.document.sections.getFirst().getHeader(Word.HeaderFooterType.primary)
;
        header.clear();
        header.insertParagraph("Public - The data is for the public and
shareable externally", "Start");
        const font = header.font;
        font.color = "#07641d";

        await context.sync();
      } else {
        const header =
context.document.sections.getFirst().getHeader(Word.HeaderFooterType.primary)
;
        header.clear();
        header.insertParagraph("Highly Confidential - The data must be secret
or in some way highly critical", "Start");
        const font = header.font;
        font.color = "#f8334d";

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

  async function registerOnParagraphChanged(event) {
    await Word.run(async (context) => {
      let eventContext =
context.document.onParagraphChanged.add(paragraphChanged);
      await context.sync();
    });
    // Calling event.completed is required. event.completed lets the platform
know that processing has completed.
    event.completed();
  }


  Office.actions.associate("changeHeader", changeHeader);
  Office.actions.associate("registerOnParagraphChanged",
registerOnParagraphChanged);
```

3. Save your changes.

Test and validate your add-in

1. Run `npm start` to build your project and launch the web server. **Ignore the Word document that is opened**.
2. Manually sideload your add-in in Word on the web by following the guidance at Sideload Office Add-ins to Office on the web. Use the **manifest.xml** in the root of the project.
3. Try opening both new and existing Word documents in Word on the web. Headers should automatically be added when they open.

See also

- Activate add-ins with events
- Debug event-based or spam-reporting add-ins
- Troubleshoot event-based and spam-reporting add-ins

Word JavaScript API overview

Article • 05/29/2025

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

- **Word JavaScript API**: These are the application-specific APIs for Word. Introduced with Office 2016, the Word JavaScript API provides strongly-typed objects that you can use to access objects and metadata in a Word document.
- **Common APIs**: The Common API, introduced with Office 2013, can be used to access features such as UI, dialogs, and client settings that are common across multiple Office applications.

This section of the documentation focuses on the Word JavaScript API, which you'll use to develop the majority of functionality in add-ins that target Word on the web, or Word 2016 and later. For information about the Common API, see Common JavaScript API object model.

Learn programming concepts

See Word JavaScript object model in Office Add-ins for information about important programming concepts.

Learn about API capabilities

Use other articles in this section of the documentation to learn how to get the whole document from an add-in, use search options in your Word add-in to find text, and more. See the table of contents for the complete list of available articles.

For hands-on experience using the Word JavaScript API to access objects in Word, complete the Word add-in tutorial.

For detailed information about the Word JavaScript API object model, see the Word JavaScript API reference documentation.

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

- Word add-ins documentation
- Word add-ins overview
- Word JavaScript API reference
- Office client application and platform availability for Office Add-ins

word package

Classes

Expand table

Word.Annotation	Represents an annotation attached to a paragraph.
Word.AnnotationCollection	Contains a collection of Word.Annotation objects.
Word.Application	Represents the application object.
Word.Bibliography	Represents the list of available sources attached to the document (in the current list) or the list of sources available in the application (in the master list).
Word.Body	Represents the body of a document or a section.
Word.Bookmark	Represents a single bookmark in a document, selection, or range. The `Bookmark` object is a member of the `Bookmark` collection. The Word.BookmarkCollection includes all the bookmarks listed in the **Bookmark** dialog box (**Insert** menu).
Word.BookmarkCollection	A collection of Word.Bookmark objects that represent the bookmarks in the specified selection, range, or document.
Word.Border	Represents the Border object for text, a paragraph, or a table.
Word.BorderCollection	Represents the collection of border styles.
Word.BorderUniversal	Represents the `BorderUniversal` object, which manages borders for a range, paragraph, table, or frame.
Word.BorderUniversalCollection	Represents the collection of Word.BorderUniversal objects.
Word.Break	Represents a break in a Word document. This could be a page, column, or section break.
Word.BreakCollection	Contains a collection of Word.Break objects.
Word.BuildingBlock	Represents a building block in a template. A building block is pre-built content, similar to autotext, that may contain text, images, and formatting.

Word.BuildingBlockCategory	Represents a category of building blocks in a Word document.
Word.BuildingBlockCategoryCollection	Represents a collection of Word.BuildingBlockCategory objects in a Word document.
Word.BuildingBlockCollection	Represents a collection of Word.BuildingBlock objects for a specific building block type and category in a template.
Word.BuildingBlockEntryCollection	Represents a collection of building block entries in a Word template.
Word.BuildingBlockGalleryContentControl	Represents the `BuildingBlockGalleryContentControl` object.
Word.BuildingBlockTypeItem	Represents a type of building block in a Word document.
Word.BuildingBlockTypeItemCollection	Represents a collection of building block types in a Word document.
Word.Canvas	Represents a canvas in the document. To get the corresponding Shape object, use Canvas.shape.
Word.CheckboxContentControl	The data specific to content controls of type CheckBox.
Word.ColorFormat	Represents the color formatting of a shape or text in Word.
Word.ComboBoxContentControl	The data specific to content controls of type 'ComboBox'.
Word.Comment	Represents a comment in the document.
Word.CommentCollection	Contains a collection of Word.Comment objects.
Word.CommentContentRange	
Word.CommentReply	Represents a comment reply in the document.
Word.CommentReplyCollection	Contains a collection of Word.CommentReply objects. Represents all comment replies in one comment thread.

Word.Content Control	Represents a content control. Content controls are bounded and potentially labeled regions in a document that serve as containers for specific types of content. Individual content controls may contain contents such as images, tables, or paragraphs of formatted text. Currently, only rich text, plain text, checkbox, dropdown list, and combo box content controls are supported.
Word.Content Control Collection	Contains a collection of Word.ContentControl objects. Content controls are bounded and potentially labeled regions in a document that serve as containers for specific types of content. Individual content controls may contain contents such as images, tables, or paragraphs of formatted text. Currently, only rich text, plain text, checkbox, dropdown list, and combo box content controls are supported.
Word.Content ControlListItem	Represents a list item in a dropdown list or combo box content control.
Word.Content ControlListItem Collection	Contains a collection of Word.ContentControlListItem objects that represent the items in a dropdown list or combo box content control.
Word.Critique Annotation	Represents an annotation wrapper around critique displayed in the document.
Word.Custom Property	Represents a custom property.
Word.Custom Property Collection	Contains the collection of Word.CustomProperty objects.
Word.Custom XmlNode	Represents an XML node in a tree in the document. The `CustomXmlNode` object is a member of the Word.CustomXmlNodeCollection object.
Word.Custom XmlNode Collection	Contains a collection of Word.CustomXmlNode objects representing the XML nodes in a document.
Word.Custom XmlPart	Represents a custom XML part.
Word.Custom XmlPart Collection	Contains the collection of Word.CustomXmlPart objects.
Word.Custom XmlPartScoped Collection	Contains the collection of Word.CustomXmlPart objects with a specific namespace.
Word.Custom XmlPrefix Mapping	Represents a `CustomXmlPrefixMapping` object.

Word.CustomXmlPrefixMappingCollection	Represents a collection of Word.CustomXmlPrefixMapping objects.
Word.CustomXmlSchema	Represents a schema in a Word.CustomXmlSchemaCollection object.
Word.CustomXmlSchemaCollection	Represents a collection of Word.CustomXmlSchema objects attached to a data stream.
Word.CustomXmlValidationError	Represents a single validation error in a Word.CustomXmlValidationErrorCollection object.
Word.CustomXmlValidationErrorCollection	Represents a collection of Word.CustomXmlValidationError objects.
Word.DatePickerContentControl	Represents the `DatePickerContentControl` object.
Word.Document	The Document object is the top level object. A Document object contains one or more sections, content controls, and the body that contains the contents of the document.
Word.DocumentCreated	The DocumentCreated object is the top level object created by Application.CreateDocument. A DocumentCreated object is a special Document object.
Word.DocumentLibraryVersion	Represents a document library version.
Word.DocumentLibraryVersionCollection	Represents the collection of Word.DocumentLibraryVersion objects.
Word.DocumentProperties	Represents document properties.
Word.DropCap	Represents a dropped capital letter in a Word document.
Word.DropDownListContentControl	The data specific to content controls of type DropDownList.
Word.Field	Represents a field.

Word.Field Collection	Contains a collection of Word.Field objects.
Word.FillFormat	Represents the fill formatting for a shape or text.
Word.Font	Represents a font.
Word.Frame	Represents a frame. The `Frame` object is a member of the Word.FrameCollection object.
Word.Frame Collection	Represents the collection of Word.Frame objects.
Word.Glow Format	Represents the glow formatting for the font used by the range of text.
Word.Group ContentControl	Represents the `GroupContentControl` object.
Word.Hyperlink	Represents a hyperlink in a Word document.
Word.Hyperlink Collection	Contains a collection of Word.Hyperlink objects.
Word.Index	Represents a single index. The `Index` object is a member of the Word.IndexCollection. The `IndexCollection` includes all the indexes in the document.
Word.Index Collection	A collection of Word.Index objects that represents all the indexes in the document.
Word.Inline Picture	Represents an inline picture.
Word.Inline Picture Collection	Contains a collection of Word.InlinePicture objects.
Word.Line Format	Represents line and arrowhead formatting. For a line, the `LineFormat` object contains formatting information for the line itself; for a shape with a border, this object contains formatting information for the shape's border.
Word.Line Numbering	Represents line numbers in the left margin or to the left of each newspaper-style column.
Word.Link Format	Represents the linking characteristics for an OLE object or picture.
Word.List	Contains a collection of Word.Paragraph objects.
Word.List Collection	Contains a collection of Word.List objects.

Word.List Format	Represents the list formatting characteristics of a range.
Word.ListItem	Represents the paragraph list item format.
Word.ListLevel	Represents a list level.
Word.ListLevel Collection	Contains a collection of Word.ListLevel objects.
Word.List Template	Represents a list template.
Word.NoteItem	Represents a footnote or endnote.
Word.NoteItem Collection	Contains a collection of Word.NoteItem objects.
Word.Ole Format	Represents the OLE characteristics (other than linking) for an OLE object, ActiveX control, or field.
Word.Page	Represents a page in the document. `Page` objects manage the page layout and content.
Word.Page Collection	Represents the collection of page.
Word.Page Setup	Represents the page setup settings for a Word document or section.
Word.Pane	Represents a window pane. The `Pane` object is a member of the pane collection. The pane collection includes all the window panes for a single window.
Word.Pane Collection	Represents the collection of pane.
Word.Paragraph	Represents a single paragraph in a selection, range, content control, or document body.
Word.Paragraph Collection	Contains a collection of Word.Paragraph objects.
Word.Paragraph Format	Represents a style of paragraph in a document.
Word.Picture ContentControl	Represents the `PictureContentControl` object.
Word.Range	Represents a contiguous area in a document.
Word.Range Collection	Contains a collection of Word.Range objects.

Word.ReflectionFormat	Represents the reflection formatting for a shape in Word.
Word.RepeatingSectionContentControl	Represents the `RepeatingSectionContentControl` object.
Word.RepeatingSectionItem	Represents a single item in a Word.RepeatingSectionContentControl.
Word.RepeatingSectionItemCollection	Represents a collection of Word.RepeatingSectionItem objects in a Word document.
Word.RequestContext	The RequestContext object facilitates requests to the Word application. Since the Office add-in and the Word application run in two different processes, the request context is required to get access to the Word object model from the add-in.
Word.Reviewer	Represents a single reviewer of a document in which changes have been tracked. The `Reviewer` object is a member of the Word.ReviewerCollection object.
Word.ReviewerCollection	A collection of Word.Reviewer objects that represents the reviewers of one or more documents. The `ReviewerCollection` object contains the names of all reviewers who have reviewed documents opened or edited on a computer.
Word.RevisionsFilter	Represents the current settings related to the display of reviewers' comments and revision marks in the document.
Word.SearchOptions	Specifies the options to be included in a search operation. To learn more about how to use search options in the Word JavaScript APIs, read Use search options to find text in your Word add-in.
Word.Section	Represents a section in a Word document.
Word.SectionCollection	Contains the collection of the document's Word.Section objects.
Word.Setting	Represents a setting of the add-in.
Word.SettingCollection	Contains the collection of Word.Setting objects.
Word.Shading	Represents the shading object.
Word.ShadingUniversal	Represents the `ShadingUniversal` object, which manages shading for a range, paragraph, frame, or table.
Word.ShadowFormat	Represents the shadow formatting for a shape or text in Word.
Word.Shape	Represents a shape in the header, footer, or document body. Currently, only the following shapes are supported: text boxes, geometric shapes, groups, pictures, and

	canvases.
Word.Shape Collection	Contains a collection of Word.Shape objects. Currently, only the following shapes are supported: text boxes, geometric shapes, groups, pictures, and canvases.
Word.ShapeFill	Represents the fill formatting of a shape object.
Word.Shape Group	Represents a shape group in the document. To get the corresponding Shape object, use ShapeGroup.shape.
Word.ShapeText Wrap	Represents all the properties for wrapping text around a shape.
Word.Source	Represents an individual source, such as a book, journal article, or interview.
Word.Source Collection	Represents a collection of Word.Source objects.
Word.Style	Represents a style in a Word document.
Word.Style Collection	Contains a collection of Word.Style objects.
Word.Table	Represents a table in a Word document.
Word.Table Border	Specifies the border style.
Word.TableCell	Represents a table cell in a Word document.
Word.TableCell Collection	Contains the collection of the document's TableCell objects.
Word.Table Collection	Contains the collection of the document's Table objects.
Word.Table Column	Represents a table column in a Word document.
Word.Table Column Collection	Represents a collection of Word.TableColumn objects in a Word document.
Word.TableRow	Represents a row in a Word document.
Word.TableRow Collection	Contains the collection of the document's TableRow objects.
Word.TableStyle	Represents the TableStyle object.
Word.TabStop	Represents a tab stop in a Word document.

Word.TabStop Collection	Represents a collection of tab stops in a Word document.
Word.Template	Represents a document template.
Word.Template Collection	Contains a collection of Word.Template objects that represent all the templates that are currently available. This collection includes open templates, templates attached to open documents, and global templates loaded in the **Templates and Add-ins** dialog box. To learn how to access this dialog in the Word UI, see Load or unload a template or add-in program.
Word.Text Column	Represents a single text column in a section.
Word.Text Column Collection	A collection of Word.TextColumn objects that represent all the columns of text in the document or a section of the document.
Word.TextFrame	Represents the text frame of a shape object.
Word.Three Dimensional Format	Represents a shape's three-dimensional formatting.
Word.Tracked Change	Represents a tracked change in a Word document.
Word.Tracked Change Collection	Contains a collection of Word.TrackedChange objects.
Word.View	Contains the view attributes (such as show all, field shading, and table gridlines) for a window or pane.
Word.Window	Represents the window that displays the document. A window can be split to contain multiple reading panes.
Word.Window Collection	Represents the collection of window objects.
Word.Xml Mapping	Represents the XML mapping on a Word.ContentControl object between custom XML and that content control. An XML mapping is a link between the text in a content control and an XML element in the custom XML data store for this document.

Interfaces

Expand table

Word.AnnotationClickedEventArgs	Holds annotation information that is passed back on annotation inserted event.
Word.AnnotationHoveredEventArgs	Holds annotation information that is passed back on annotation hovered event.
Word.AnnotationInsertedEventArgs	Holds annotation information that is passed back on annotation added event.
Word.AnnotationPopupActionEventArgs	Represents action information that's passed back on annotation pop-up action event.
Word.AnnotationRemovedEventArgs	Holds annotation information that is passed back on annotation removed event.
Word.AnnotationSet	Annotations set produced by the add-in. Currently supporting only critiques.
Word.CommentDetail	A structure for the ID and reply IDs of this comment.
Word.CommentEventArgs	Provides information about the comments that raised the comment event.
Word.ContentControlAddedEventArgs	Provides information about the content control that raised contentControlAdded event.
Word.ContentControlDataChangedEventArgs	Provides information about the content control that raised contentControlDataChanged event.
Word.ContentControlDeletedEventArgs	Provides information about the content control that raised contentControlDeleted event.
Word.ContentControlEnteredEventArgs	Provides information about the content control that raised contentControlEntered event.
Word.ContentControlExitedEventArgs	Provides information about the content control that raised contentControlExited event.
Word.ContentControlOptions	Specifies the options that define which content controls are returned.

Word.ContentControlPlaceholderOptions	The options that define what placeholder to be used in the content control.
Word.ContentControlSelectionChangedEventArgs	Provides information about the content control that raised contentControlSelectionChanged event.
Word.Critique	Critique that will be rendered as underline for the specified part of paragraph in the document.
Word.CritiquePopupOptions	Properties defining the behavior of the pop-up menu for a given critique.
Word.CustomXmlAddNodeOptions	The options for adding a node to the XML tree.
Word.CustomXmlAddSchemaOptions	Adds one or more schemas to a schema collection that can then be added to a stream in the data store and to the schema library.
Word.CustomXmlAddValidationErrorOptions	The options that define the descriptive error text and the state of `clearedOnUpdate`.
Word.CustomXmlAppendChildNodeOptions	The options that define the prefix mapping and the source of the custom XML data.
Word.CustomXmlInsertNodeBeforeOptions	Inserts a new node just before the context node in the tree.
Word.CustomXmlInsertSubtreeBeforeOptions	Inserts a new node just before the context node in the tree.
Word.CustomXmlReplaceChildNodeOptions	Removes the specified child node and replaces it with a different node in the same location.
Word.DocumentCompareOptions	Specifies the options to be included in a compare document operation.
Word.GetTextOptions	Specifies the options to be included in a getText operation.
Word.HyperlinkAddOptions	Specifies the options for adding to a Word.HyperlinkCollection object.

Word.IndexAddOptions	Represents options for creating an index in a Word document.
Word.IndexMarkAllEntriesOptions	Represents options for marking all index entries in a Word document.
Word.IndexMarkEntryOptions	Represents options for marking an index entry in a Word document.
Word.InsertFileOptions	Specifies the options to determine what to copy when inserting a file.
Word.InsertShapeOptions	Specifies the options to determine location and size when inserting a shape.
Word.Interfaces.AnnotationCollectionData	An interface describing the data returned by calling `annotationCollection.toJSON()`.
Word.Interfaces.AnnotationCollectionLoadOptions	Contains a collection of Word.Annotation objects.
Word.Interfaces.AnnotationCollectionUpdateData	An interface for updating data on the `AnnotationCollection` object, for use in `annotationCollection.set({ ... })`.
Word.Interfaces.AnnotationData	An interface describing the data returned by calling `annotation.toJSON()`.
Word.Interfaces.AnnotationLoadOptions	Represents an annotation attached to a paragraph.
Word.Interfaces.ApplicationData	An interface describing the data returned by calling `application.toJSON()`.
Word.Interfaces.ApplicationLoadOptions	Represents the application object.
Word.Interfaces.ApplicationUpdateData	An interface for updating data on the `Application` object, for use in `application.set({ ... })`.
Word.Interfaces.BibliographyData	An interface describing the data returned by calling `bibliography.toJSON()`.

Word.Interfaces.BibliographyLoadOptions	Represents the list of available sources attached to the document (in the current list) or the list of sources available in the application (in the master list).
Word.Interfaces.BibliographyUpdateData	An interface for updating data on the `Bibliography` object, for use in `bibliography.set({ ... })`.
Word.Interfaces.BodyData	An interface describing the data returned by calling `body.toJSON()`.
Word.Interfaces.BodyLoadOptions	Represents the body of a document or a section.
Word.Interfaces.BodyUpdateData	An interface for updating data on the `Body` object, for use in `body.set({ ... })`.
Word.Interfaces.BookmarkCollectionData	An interface describing the data returned by calling `bookmarkCollection.toJSON()`.
Word.Interfaces.BookmarkCollectionLoadOptions	A collection of Word.Bookmark objects that represent the bookmarks in the specified selection, range, or document.
Word.Interfaces.BookmarkCollectionUpdateData	An interface for updating data on the `BookmarkCollection` object, for use in `bookmarkCollection.set({ ... })`.
Word.Interfaces.BookmarkData	An interface describing the data returned by calling `bookmark.toJSON()`.
Word.Interfaces.BookmarkLoadOptions	Represents a single bookmark in a document, selection, or range. The `Bookmark` object is a member of the `Bookmark` collection. The Word.BookmarkCollection includes all the bookmarks listed in the **Bookmark** dialog box (**Insert** menu).
Word.Interfaces.BookmarkUpdateData	An interface for updating data on the `Bookmark` object, for use in `bookmark.set({ ... })`.
Word.Interfaces.BorderCollectionData	An interface describing the data returned by calling `borderCollection.toJSON()`.
Word.Interfaces.BorderCollectionLoadOptions	Represents the collection of border styles.

Word.Interfaces.BorderCollectionUpdateData	An interface for updating data on the `BorderCollection` object, for use in `borderCollection.set({ ... })`.
Word.Interfaces.BorderData	An interface describing the data returned by calling `border.toJSON()`.
Word.Interfaces.BorderLoadOptions	Represents the Border object for text, a paragraph, or a table.
Word.Interfaces.BorderUniversalCollectionData	An interface describing the data returned by calling `borderUniversalCollection.toJSON()`.
Word.Interfaces.BorderUniversalCollectionLoadOptions	Represents the collection of Word.BorderUniversal objects.
Word.Interfaces.BorderUniversalCollectionUpdateData	An interface for updating data on the `BorderUniversalCollection` object, for use in `borderUniversalCollection.set({ ... })`.
Word.Interfaces.BorderUniversalData	An interface describing the data returned by calling `borderUniversal.toJSON()`.
Word.Interfaces.BorderUniversalLoadOptions	Represents the `BorderUniversal` object, which manages borders for a range, paragraph, table, or frame.
Word.Interfaces.BorderUniversalUpdateData	An interface for updating data on the `BorderUniversal` object, for use in `borderUniversal.set({ ... })`.
Word.Interfaces.BorderUpdateData	An interface for updating data on the `Border` object, for use in `border.set({ ... })`.
Word.Interfaces.BreakCollectionData	An interface describing the data returned by calling `breakCollection.toJSON()`.
Word.Interfaces.BreakCollectionLoadOptions	Contains a collection of Word.Break objects.
Word.Interfaces.BreakCollection	An interface for updating data on the `BreakCollection` object, for use in `breakCollection.set({ ... })`.

UpdateData	
Word.Interfaces.BreakData	An interface describing the data returned by calling `break.toJSON()`.
Word.Interfaces.BreakLoadOptions	Represents a break in a Word document.
Word.Interfaces.BreakUpdateData	An interface for updating data on the `Break` object, for use in `break.set({ ... })`.
Word.Interfaces.BuildingBlockCategoryData	An interface describing the data returned by calling `buildingBlockCategory.toJSON()`.
Word.Interfaces.BuildingBlockCategoryLoadOptions	Represents a category of building blocks in a Word document.
Word.Interfaces.BuildingBlockData	An interface describing the data returned by calling `buildingBlock.toJSON()`.
Word.Interfaces.BuildingBlockGalleryContentControlData	An interface describing the data returned by calling `buildingBlockGalleryContentControl.toJSON()`.
Word.Interfaces.BuildingBlockGalleryContentControlLoadOptions	Represents the `BuildingBlockGalleryContentControl` object.
Word.Interfaces.BuildingBlockGalleryContentControlUpdateData	An interface for updating data on the `BuildingBlockGalleryContentControl` object, for use in `buildingBlockGalleryContentControl.set({ ... })`.
Word.Interfaces.BuildingBlockLoadOptions	Represents a building block in a template. A building block is pre-built content, similar to autotext, that may contain text, images, and formatting.
Word.Interfaces.BuildingBlockTypeItemData	An interface describing the data returned by calling `buildingBlockTypeItem.toJSON()`.
Word.Interfaces.BuildingBlockType	Represents a type of building block in a Word document.

ItemLoadOptions	
Word.Interfaces.BuildingBlockUpdateData	An interface for updating data on the `BuildingBlock` object, for use in `buildingBlock.set({ ... })`.
Word.Interfaces.CanvasData	An interface describing the data returned by calling `canvas.toJSON()`.
Word.Interfaces.CanvasLoadOptions	Represents a canvas in the document. To get the corresponding Shape object, use Canvas.shape.
Word.Interfaces.CanvasUpdateData	An interface for updating data on the `Canvas` object, for use in `canvas.set({ ... })`.
Word.Interfaces.CheckboxContentControlData	An interface describing the data returned by calling `checkboxContentControl.toJSON()`.
Word.Interfaces.CheckboxContentControlLoadOptions	The data specific to content controls of type CheckBox.
Word.Interfaces.CheckboxContentControlUpdateData	An interface for updating data on the `CheckboxContentControl` object, for use in `checkboxContentControl.set({ ... })`.
Word.Interfaces.CollectionLoadOptions	Provides ways to load properties of only a subset of members of a collection.
Word.Interfaces.ColorFormatData	An interface describing the data returned by calling `colorFormat.toJSON()`.
Word.Interfaces.ColorFormatLoadOptions	Represents the color formatting of a shape or text in Word.
Word.Interfaces.ColorFormatUpdateData	An interface for updating data on the `ColorFormat` object, for use in `colorFormat.set({ ... })`.
Word.Interfaces.ComboBoxContentControlData	An interface describing the data returned by calling `comboBoxContentControl.toJSON()`.

Word.Interfaces.CommentCollectionData	An interface describing the data returned by calling `commentCollection.toJSON()`.
Word.Interfaces.CommentCollectionLoadOptions	Contains a collection of Word.Comment objects.
Word.Interfaces.CommentCollectionUpdateData	An interface for updating data on the `CommentCollection` object, for use in `commentCollection.set({ ... })`.
Word.Interfaces.CommentContentRangeData	An interface describing the data returned by calling `commentContentRange.toJSON()`.
Word.Interfaces.CommentContentRangeLoadOptions	
Word.Interfaces.CommentContentRangeUpdateData	An interface for updating data on the `CommentContentRange` object, for use in `commentContentRange.set({ ... })`.
Word.Interfaces.CommentData	An interface describing the data returned by calling `comment.toJSON()`.
Word.Interfaces.CommentLoadOptions	Represents a comment in the document.
Word.Interfaces.CommentReplyCollectionData	An interface describing the data returned by calling `commentReplyCollection.toJSON()`.
Word.Interfaces.CommentReplyCollectionLoadOptions	Contains a collection of Word.CommentReply objects. Represents all comment replies in one comment thread.
Word.Interfaces.CommentReplyCollectionUpdateData	An interface for updating data on the `CommentReplyCollection` object, for use in `commentReplyCollection.set({ ... })`.
Word.Interfaces.CommentReplyData	An interface describing the data returned by calling `commentReply.toJSON()`.
Word.Interfaces.CommentReply	Represents a comment reply in the document.

LoadOptions	
Word.Interfaces.CommentReplyUpdateData	An interface for updating data on the `CommentReply` object, for use in `commentReply.set({ ... })`.
Word.Interfaces.CommentUpdateData	An interface for updating data on the `Comment` object, for use in `comment.set({ ... })`.
Word.Interfaces.ContentControlCollectionData	An interface describing the data returned by calling `contentControlCollection.toJSON()`.
Word.Interfaces.ContentControlCollectionLoadOptions	Contains a collection of Word.ContentControl objects. Content controls are bounded and potentially labeled regions in a document that serve as containers for specific types of content. Individual content controls may contain contents such as images, tables, or paragraphs of formatted text. Currently, only rich text, plain text, checkbox, dropdown list, and combo box content controls are supported.
Word.Interfaces.ContentControlCollectionUpdateData	An interface for updating data on the `ContentControlCollection` object, for use in `contentControlCollection.set({ ... })`.
Word.Interfaces.ContentControlData	An interface describing the data returned by calling `contentControl.toJSON()`.
Word.Interfaces.ContentControlListItemCollectionData	An interface describing the data returned by calling `contentControlListItemCollection.toJSON()`.
Word.Interfaces.ContentControlListItemCollectionLoadOptions	Contains a collection of Word.ContentControlListItem objects that represent the items in a dropdown list or combo box content control.
Word.Interfaces.ContentControlListItemCollectionUpdateData	An interface for updating data on the `ContentControlListItemCollection` object, for use in `contentControlListItemCollection.set({ ... })`.
Word.Interfaces.ContentControlListItemData	An interface describing the data returned by calling `contentControlListItem.toJSON()`.
Word.Interfaces.ContentControl	Represents a list item in a dropdown list or combo box content control.

ListItemLoad Options	
Word.Interfaces. ContentControl ListItemUpdate Data	An interface for updating data on the `ContentControlListItem` object, for use in `contentControlListItem.set({ ... })`.
Word.Interfaces. ContentControl LoadOptions	Represents a content control. Content controls are bounded and potentially labeled regions in a document that serve as containers for specific types of content. Individual content controls may contain contents such as images, tables, or paragraphs of formatted text. Currently, only rich text, plain text, checkbox, dropdown list, and combo box content controls are supported.
Word.Interfaces. ContentControl UpdateData	An interface for updating data on the `ContentControl` object, for use in `contentControl.set({ ... })`.
Word.Interfaces. Critique AnnotationData	An interface describing the data returned by calling `critiqueAnnotation.toJSON()`.
Word.Interfaces. Critique AnnotationLoad Options	Represents an annotation wrapper around critique displayed in the document.
Word.Interfaces. CustomProperty CollectionData	An interface describing the data returned by calling `customPropertyCollection.toJSON()`.
Word.Interfaces. CustomProperty CollectionLoad Options	Contains the collection of Word.CustomProperty objects.
Word.Interfaces. CustomProperty CollectionUpdate Data	An interface for updating data on the `CustomPropertyCollection` object, for use in `customPropertyCollection.set({ ... })`.
Word.Interfaces. CustomProperty Data	An interface describing the data returned by calling `customProperty.toJSON()`.
Word.Interfaces. CustomProperty LoadOptions	Represents a custom property.
Word.Interfaces. CustomProperty	An interface for updating data on the `CustomProperty` object, for use in `customProperty.set({ ... })`.

UpdateData	
Word.Interfaces.CustomXmlNodeCollectionData	An interface describing the data returned by calling `customXmlNodeCollection.toJSON()`.
Word.Interfaces.CustomXmlNodeCollectionLoadOptions	Contains a collection of Word.CustomXmlNode objects representing the XML nodes in a document.
Word.Interfaces.CustomXmlNodeCollectionUpdateData	An interface for updating data on the `CustomXmlNodeCollection` object, for use in `customXmlNodeCollection.set({ ... })`.
Word.Interfaces.CustomXmlNodeData	An interface describing the data returned by calling `customXmlNode.toJSON()`.
Word.Interfaces.CustomXmlNodeLoadOptions	Represents an XML node in a tree in the document. The `CustomXmlNode` object is a member of the Word.CustomXmlNodeCollection object.
Word.Interfaces.CustomXmlNodeUpdateData	An interface for updating data on the `CustomXmlNode` object, for use in `customXmlNode.set({ ... })`.
Word.Interfaces.CustomXmlPartCollectionData	An interface describing the data returned by calling `customXmlPartCollection.toJSON()`.
Word.Interfaces.CustomXmlPartCollectionLoadOptions	Contains the collection of Word.CustomXmlPart objects.
Word.Interfaces.CustomXmlPartCollectionUpdateData	An interface for updating data on the `CustomXmlPartCollection` object, for use in `customXmlPartCollection.set({ ... })`.
Word.Interfaces.CustomXmlPartData	An interface describing the data returned by calling `customXmlPart.toJSON()`.
Word.Interfaces.CustomXmlPartLoadOptions	Represents a custom XML part.

Word.Interfaces.CustomXmlPartScopedCollectionData	An interface describing the data returned by calling `customXmlPartScopedCollection.toJSON()`.
Word.Interfaces.CustomXmlPartScopedCollectionLoadOptions	Contains the collection of Word.CustomXmlPart objects with a specific namespace.
Word.Interfaces.CustomXmlPartScopedCollectionUpdateData	An interface for updating data on the `CustomXmlPartScopedCollection` object, for use in `customXmlPartScopedCollection.set({ ... })`.
Word.Interfaces.CustomXmlPartUpdateData	An interface for updating data on the `CustomXmlPart` object, for use in `customXmlPart.set({ ... })`.
Word.Interfaces.CustomXmlPrefixMappingCollectionData	An interface describing the data returned by calling `customXmlPrefixMappingCollection.toJSON()`.
Word.Interfaces.CustomXmlPrefixMappingCollectionLoadOptions	Represents a collection of Word.CustomXmlPrefixMapping objects.
Word.Interfaces.CustomXmlPrefixMappingCollectionUpdateData	An interface for updating data on the `CustomXmlPrefixMappingCollection` object, for use in `customXmlPrefixMappingCollection.set({ ... })`.
Word.Interfaces.CustomXmlPrefixMappingData	An interface describing the data returned by calling `customXmlPrefixMapping.toJSON()`.
Word.Interfaces.CustomXmlPrefixMappingLoadOptions	Represents a `CustomXmlPrefixMapping` object.
Word.Interfaces.CustomXmlSchemaCollectionData	An interface describing the data returned by calling `customXmlSchemaCollection.toJSON()`.

Word.Interfaces.CustomXmlSchemaCollectionLoadOptions	Represents a collection of Word.CustomXmlSchema objects attached to a data stream.
Word.Interfaces.CustomXmlSchemaCollectionUpdateData	An interface for updating data on the `CustomXmlSchemaCollection` object, for use in `customXmlSchemaCollection.set({ ... })`.
Word.Interfaces.CustomXmlSchemaData	An interface describing the data returned by calling `customXmlSchema.toJSON()`.
Word.Interfaces.CustomXmlSchemaLoadOptions	Represents a schema in a Word.CustomXmlSchemaCollection object.
Word.Interfaces.CustomXmlValidationErrorCollectionData	An interface describing the data returned by calling `customXmlValidationErrorCollection.toJSON()`.
Word.Interfaces.CustomXmlValidationErrorCollectionLoadOptions	Represents a collection of Word.CustomXmlValidationError objects.
Word.Interfaces.CustomXmlValidationErrorCollectionUpdateData	An interface for updating data on the `CustomXmlValidationErrorCollection` object, for use in `customXmlValidationErrorCollection.set({ ... })`.
Word.Interfaces.CustomXmlValidationErrorData	An interface describing the data returned by calling `customXmlValidationError.toJSON()`.
Word.Interfaces.CustomXmlValidationErrorLoadOptions	Represents a single validation error in a Word.CustomXmlValidationErrorCollection object.
Word.Interfaces.CustomXmlValidationErrorUpdateData	An interface for updating data on the `CustomXmlValidationError` object, for use in `customXmlValidationError.set({ ... })`.

Word.Interfaces. DatePicker ContentControl Data	An interface describing the data returned by calling `datePickerContentControl.toJSON()`.
Word.Interfaces. DatePicker ContentControl LoadOptions	Represents the `DatePickerContentControl` object.
Word.Interfaces. DatePicker ContentControl UpdateData	An interface for updating data on the `DatePickerContentControl` object, for use in `datePickerContentControl.set({ ... })`.
Word.Interfaces. DocumentCreated Data	An interface describing the data returned by calling `documentCreated.toJSON()`.
Word.Interfaces. DocumentCreated LoadOptions	The DocumentCreated object is the top level object created by Application.CreateDocument. A DocumentCreated object is a special Document object.
Word.Interfaces. DocumentCreated UpdateData	An interface for updating data on the `DocumentCreated` object, for use in `documentCreated.set({ ... })`.
Word.Interfaces. DocumentData	An interface describing the data returned by calling `document.toJSON()`.
Word.Interfaces. DocumentLibrary VersionCollection Data	An interface describing the data returned by calling `documentLibraryVersionCollection.toJSON()`.
Word.Interfaces. DocumentLibrary VersionCollection LoadOptions	Represents the collection of Word.DocumentLibraryVersion objects.
Word.Interfaces. DocumentLibrary VersionCollection UpdateData	An interface for updating data on the `DocumentLibraryVersionCollection` object, for use in `documentLibraryVersionCollection.set({ ... })`.
Word.Interfaces. DocumentLibrary VersionData	An interface describing the data returned by calling `documentLibraryVersion.toJSON()`.
Word.Interfaces. DocumentLibrary	Represents a document library version.

VersionLoad Options	
Word.Interfaces. DocumentLoad Options	The Document object is the top level object. A Document object contains one or more sections, content controls, and the body that contains the contents of the document.
Word.Interfaces. Document PropertiesData	An interface describing the data returned by calling `documentProperties.toJSON()`.
Word.Interfaces. Document PropertiesLoad Options	Represents document properties.
Word.Interfaces. Document PropertiesUpdate Data	An interface for updating data on the `DocumentProperties` object, for use in `documentProperties.set({ ... })`.
Word.Interfaces. DocumentUpdate Data	An interface for updating data on the `Document` object, for use in `document.set({ ... })`.
Word.Interfaces. DropCapData	An interface describing the data returned by calling `dropCap.toJSON()`.
Word.Interfaces. DropCapLoad Options	Represents a dropped capital letter in a Word document.
Word.Interfaces. DropDownList ContentControl Data	An interface describing the data returned by calling `dropDownListContentControl.toJSON()`.
Word.Interfaces. FieldCollection Data	An interface describing the data returned by calling `fieldCollection.toJSON()`.
Word.Interfaces. FieldCollection LoadOptions	Contains a collection of Word.Field objects.
Word.Interfaces. FieldCollection UpdateData	An interface for updating data on the `FieldCollection` object, for use in `fieldCollection.set({ ... })`.
Word.Interfaces. FieldData	An interface describing the data returned by calling `field.toJSON()`.

Word.Interfaces.FieldLoadOptions	Represents a field.
Word.Interfaces.FieldUpdateData	An interface for updating data on the `Field` object, for use in `field.set({ ... })`.
Word.Interfaces.FillFormatData	An interface describing the data returned by calling `fillFormat.toJSON()`.
Word.Interfaces.FillFormatLoadOptions	Represents the fill formatting for a shape or text.
Word.Interfaces.FillFormatUpdateData	An interface for updating data on the `FillFormat` object, for use in `fillFormat.set({ ... })`.
Word.Interfaces.FontData	An interface describing the data returned by calling `font.toJSON()`.
Word.Interfaces.FontLoadOptions	Represents a font.
Word.Interfaces.FontUpdateData	An interface for updating data on the `Font` object, for use in `font.set({ ... })`.
Word.Interfaces.FrameCollectionData	An interface describing the data returned by calling `frameCollection.toJSON()`.
Word.Interfaces.FrameCollectionLoadOptions	Represents the collection of Word.Frame objects.
Word.Interfaces.FrameCollectionUpdateData	An interface for updating data on the `FrameCollection` object, for use in `frameCollection.set({ ... })`.
Word.Interfaces.FrameData	An interface describing the data returned by calling `frame.toJSON()`.
Word.Interfaces.FrameLoadOptions	Represents a frame. The `Frame` object is a member of the Word.FrameCollection object.
Word.Interfaces.FrameUpdateData	An interface for updating data on the `Frame` object, for use in `frame.set({ ... })`.
Word.Interfaces.GlowFormatData	An interface describing the data returned by calling `glowFormat.toJSON()`.

Word.Interfaces.GlowFormatLoadOptions	Represents the glow formatting for the font used by the range of text.
Word.Interfaces.GlowFormatUpdateData	An interface for updating data on the `GlowFormat` object, for use in `glowFormat.set({ ... })`.
Word.Interfaces.GroupContentControlData	An interface describing the data returned by calling `groupContentControl.toJSON()`.
Word.Interfaces.GroupContentControlLoadOptions	Represents the `GroupContentControl` object.
Word.Interfaces.GroupContentControlUpdateData	An interface for updating data on the `GroupContentControl` object, for use in `groupContentControl.set({ ... })`.
Word.Interfaces.HyperlinkCollectionData	An interface describing the data returned by calling `hyperlinkCollection.toJSON()`.
Word.Interfaces.HyperlinkCollectionLoadOptions	Contains a collection of Word.Hyperlink objects.
Word.Interfaces.HyperlinkCollectionUpdateData	An interface for updating data on the `HyperlinkCollection` object, for use in `hyperlinkCollection.set({ ... })`.
Word.Interfaces.HyperlinkData	An interface describing the data returned by calling `hyperlink.toJSON()`.
Word.Interfaces.HyperlinkLoadOptions	Represents a hyperlink in a Word document.
Word.Interfaces.HyperlinkUpdateData	An interface for updating data on the `Hyperlink` object, for use in `hyperlink.set({ ... })`.
Word.Interfaces.IndexCollectionData	An interface describing the data returned by calling `indexCollection.toJSON()`.

Word.Interfaces. IndexCollection LoadOptions	A collection of Word.Index objects that represents all the indexes in the document.
Word.Interfaces. IndexCollection UpdateData	An interface for updating data on the `IndexCollection` object, for use in `indexCollection.set({ ... })`.
Word.Interfaces. IndexData	An interface describing the data returned by calling `index.toJSON()`.
Word.Interfaces. IndexLoadOptions	Represents a single index. The `Index` object is a member of the Word.IndexCollection. The `IndexCollection` includes all the indexes in the document.
Word.Interfaces. IndexUpdateData	An interface for updating data on the `Index` object, for use in `index.set({ ... })`.
Word.Interfaces. InlinePicture CollectionData	An interface describing the data returned by calling `inlinePictureCollection.toJSON()`.
Word.Interfaces. InlinePicture CollectionLoad Options	Contains a collection of Word.InlinePicture objects.
Word.Interfaces. InlinePicture CollectionUpdate Data	An interface for updating data on the `InlinePictureCollection` object, for use in `inlinePictureCollection.set({ ... })`.
Word.Interfaces. InlinePictureData	An interface describing the data returned by calling `inlinePicture.toJSON()`.
Word.Interfaces. InlinePictureLoad Options	Represents an inline picture.
Word.Interfaces. InlinePicture UpdateData	An interface for updating data on the `InlinePicture` object, for use in `inlinePicture.set({ ... })`.
Word.Interfaces. LineFormatData	An interface describing the data returned by calling `lineFormat.toJSON()`.
Word.Interfaces. LineFormatLoad Options	Represents line and arrowhead formatting. For a line, the `LineFormat` object contains formatting information for the line itself; for a shape with a border, this object contains formatting information for the shape's border.
Word.Interfaces. LineFormat	An interface for updating data on the `LineFormat` object, for use in `lineFormat.set({ ... })`.

UpdateData	
Word.Interfaces.LineNumberingData	An interface describing the data returned by calling `lineNumbering.toJSON()`.
Word.Interfaces.LineNumberingLoadOptions	Represents line numbers in the left margin or to the left of each newspaper-style column.
Word.Interfaces.LineNumberingUpdateData	An interface for updating data on the `LineNumbering` object, for use in `lineNumbering.set({ ... })`.
Word.Interfaces.LinkFormatData	An interface describing the data returned by calling `linkFormat.toJSON()`.
Word.Interfaces.LinkFormatLoadOptions	Represents the linking characteristics for an OLE object or picture.
Word.Interfaces.LinkFormatUpdateData	An interface for updating data on the `LinkFormat` object, for use in `linkFormat.set({ ... })`.
Word.Interfaces.ListCollectionData	An interface describing the data returned by calling `listCollection.toJSON()`.
Word.Interfaces.ListCollectionLoadOptions	Contains a collection of Word.List objects.
Word.Interfaces.ListCollectionUpdateData	An interface for updating data on the `ListCollection` object, for use in `listCollection.set({ ... })`.
Word.Interfaces.ListData	An interface describing the data returned by calling `list.toJSON()`.
Word.Interfaces.ListFormatData	An interface describing the data returned by calling `listFormat.toJSON()`.
Word.Interfaces.ListFormatLoadOptions	Represents the list formatting characteristics of a range.
Word.Interfaces.ListFormatUpdateData	An interface for updating data on the `ListFormat` object, for use in `listFormat.set({ ... })`.
Word.Interfaces.ListItemData	An interface describing the data returned by calling `listItem.toJSON()`.

Word.Interfaces.ListItemLoadOptions	Represents the paragraph list item format.
Word.Interfaces.ListItemUpdateData	An interface for updating data on the `ListItem` object, for use in `listItem.set({ ... })`.
Word.Interfaces.ListLevelCollectionData	An interface describing the data returned by calling `listLevelCollection.toJSON()`.
Word.Interfaces.ListLevelCollectionLoadOptions	Contains a collection of Word.ListLevel objects.
Word.Interfaces.ListLevelCollectionUpdateData	An interface for updating data on the `ListLevelCollection` object, for use in `listLevelCollection.set({ ... })`.
Word.Interfaces.ListLevelData	An interface describing the data returned by calling `listLevel.toJSON()`.
Word.Interfaces.ListLevelLoadOptions	Represents a list level.
Word.Interfaces.ListLevelUpdateData	An interface for updating data on the `ListLevel` object, for use in `listLevel.set({ ... })`.
Word.Interfaces.ListLoadOptions	Contains a collection of Word.Paragraph objects.
Word.Interfaces.ListTemplateData	An interface describing the data returned by calling `listTemplate.toJSON()`.
Word.Interfaces.ListTemplateLoadOptions	Represents a list template.
Word.Interfaces.ListTemplateUpdateData	An interface for updating data on the `ListTemplate` object, for use in `listTemplate.set({ ... })`.
Word.Interfaces.NoteItemCollectionData	An interface describing the data returned by calling `noteItemCollection.toJSON()`.

Word.Interfaces.NoteItemCollectionLoadOptions	Contains a collection of Word.NoteItem objects.
Word.Interfaces.NoteItemCollectionUpdateData	An interface for updating data on the `NoteItemCollection` object, for use in `noteItemCollection.set({ ... })`.
Word.Interfaces.NoteItemData	An interface describing the data returned by calling `noteItem.toJSON()`.
Word.Interfaces.NoteItemLoadOptions	Represents a footnote or endnote.
Word.Interfaces.NoteItemUpdateData	An interface for updating data on the `NoteItem` object, for use in `noteItem.set({ ... })`.
Word.Interfaces.OleFormatData	An interface describing the data returned by calling `oleFormat.toJSON()`.
Word.Interfaces.OleFormatLoadOptions	Represents the OLE characteristics (other than linking) for an OLE object, ActiveX control, or field.
Word.Interfaces.OleFormatUpdateData	An interface for updating data on the `OleFormat` object, for use in `oleFormat.set({ ... })`.
Word.Interfaces.PageCollectionData	An interface describing the data returned by calling `pageCollection.toJSON()`.
Word.Interfaces.PageCollectionLoadOptions	Represents the collection of page.
Word.Interfaces.PageCollectionUpdateData	An interface for updating data on the `PageCollection` object, for use in `pageCollection.set({ ... })`.
Word.Interfaces.PageData	An interface describing the data returned by calling `page.toJSON()`.
Word.Interfaces.PageLoadOptions	Represents a page in the document. `Page` objects manage the page layout and content.
Word.Interfaces.	An interface describing the data returned by calling `pageSetup.toJSON()`.

PageSetupData	
Word.Interfaces. PageSetupLoad Options	Represents the page setup settings for a Word document or section.
Word.Interfaces. PageSetupUpdate Data	An interface for updating data on the `PageSetup` object, for use in `pageSetup.set({ ... })`.
Word.Interfaces. PaneCollection Data	An interface describing the data returned by calling `paneCollection.toJSON()`.
Word.Interfaces. PaneCollection UpdateData	An interface for updating data on the `PaneCollection` object, for use in `paneCollection.set({ ... })`.
Word.Interfaces. PaneData	An interface describing the data returned by calling `pane.toJSON()`.
Word.Interfaces. Paragraph CollectionData	An interface describing the data returned by calling `paragraphCollection.toJSON()`.
Word.Interfaces. Paragraph CollectionLoad Options	Contains a collection of Word.Paragraph objects.
Word.Interfaces. Paragraph CollectionUpdate Data	An interface for updating data on the `ParagraphCollection` object, for use in `paragraphCollection.set({ ... })`.
Word.Interfaces. ParagraphData	An interface describing the data returned by calling `paragraph.toJSON()`.
Word.Interfaces. ParagraphFormat Data	An interface describing the data returned by calling `paragraphFormat.toJSON()`.
Word.Interfaces. ParagraphFormat LoadOptions	Represents a style of paragraph in a document.
Word.Interfaces. ParagraphFormat UpdateData	An interface for updating data on the `ParagraphFormat` object, for use in `paragraphFormat.set({ ... })`.
Word.Interfaces. ParagraphLoad	Represents a single paragraph in a selection, range, content control, or document body.

Options	
Word.Interfaces.ParagraphUpdateData	An interface for updating data on the `Paragraph` object, for use in `paragraph.set({ ... })`.
Word.Interfaces.PictureContentControlData	An interface describing the data returned by calling `pictureContentControl.toJSON()`.
Word.Interfaces.PictureContentControlLoadOptions	Represents the `PictureContentControl` object.
Word.Interfaces.PictureContentControlUpdateData	An interface for updating data on the `PictureContentControl` object, for use in `pictureContentControl.set({ ... })`.
Word.Interfaces.RangeCollectionData	An interface describing the data returned by calling `rangeCollection.toJSON()`.
Word.Interfaces.RangeCollectionLoadOptions	Contains a collection of Word.Range objects.
Word.Interfaces.RangeCollectionUpdateData	An interface for updating data on the `RangeCollection` object, for use in `rangeCollection.set({ ... })`.
Word.Interfaces.RangeData	An interface describing the data returned by calling `range.toJSON()`.
Word.Interfaces.RangeLoadOptions	Represents a contiguous area in a document.
Word.Interfaces.RangeUpdateData	An interface for updating data on the `Range` object, for use in `range.set({ ... })`.
Word.Interfaces.ReflectionFormatData	An interface describing the data returned by calling `reflectionFormat.toJSON()`.
Word.Interfaces.ReflectionFormatLoadOptions	Represents the reflection formatting for a shape in Word.

Word.Interfaces.ReflectionFormatUpdateData	An interface for updating data on the `ReflectionFormat` object, for use in `reflectionFormat.set({ ... })`.
Word.Interfaces.RepeatingSectionContentControlData	An interface describing the data returned by calling `repeatingSectionContentControl.toJSON()`.
Word.Interfaces.RepeatingSectionContentControlLoadOptions	Represents the `RepeatingSectionContentControl` object.
Word.Interfaces.RepeatingSectionContentControlUpdateData	An interface for updating data on the `RepeatingSectionContentControl` object, for use in `repeatingSectionContentControl.set({ ... })`.
Word.Interfaces.RepeatingSectionItemData	An interface describing the data returned by calling `repeatingSectionItem.toJSON()`.
Word.Interfaces.RepeatingSectionItemLoadOptions	Represents a single item in a Word.RepeatingSectionContentControl.
Word.Interfaces.RepeatingSectionItemUpdateData	An interface for updating data on the `RepeatingSectionItem` object, for use in `repeatingSectionItem.set({ ... })`.
Word.Interfaces.ReviewerCollectionData	An interface describing the data returned by calling `reviewerCollection.toJSON()`.
Word.Interfaces.ReviewerCollectionLoadOptions	A collection of Word.Reviewer objects that represents the reviewers of one or more documents. The `ReviewerCollection` object contains the names of all reviewers who have reviewed documents opened or edited on a computer.
Word.Interfaces.ReviewerCollectionUpdateData	An interface for updating data on the `ReviewerCollection` object, for use in `reviewerCollection.set({ ... })`.
Word.Interfaces.ReviewerData	An interface describing the data returned by calling `reviewer.toJSON()`.
Word.Interfaces.ReviewerLoadOptions	Represents a single reviewer of a document in which changes have been tracked. The `Reviewer` object is a member of the Word.ReviewerCollection object.

Word.Interfaces.ReviewerUpdateData	An interface for updating data on the `Reviewer` object, for use in `reviewer.set({ ... })`.
Word.Interfaces.RevisionsFilterData	An interface describing the data returned by calling `revisionsFilter.toJSON()`.
Word.Interfaces.RevisionsFilterLoadOptions	Represents the current settings related to the display of reviewers' comments and revision marks in the document.
Word.Interfaces.RevisionsFilterUpdateData	An interface for updating data on the `RevisionsFilter` object, for use in `revisionsFilter.set({ ... })`.
Word.Interfaces.SearchOptionsData	An interface describing the data returned by calling `searchOptions.toJSON()`.
Word.Interfaces.SearchOptionsLoadOptions	Specifies the options to be included in a search operation. To learn more about how to use search options in the Word JavaScript APIs, read Use search options to find text in your Word add-in.
Word.Interfaces.SearchOptionsUpdateData	An interface for updating data on the `SearchOptions` object, for use in `searchOptions.set({ ... })`.
Word.Interfaces.SectionCollectionData	An interface describing the data returned by calling `sectionCollection.toJSON()`.
Word.Interfaces.SectionCollectionLoadOptions	Contains the collection of the document's Word.Section objects.
Word.Interfaces.SectionCollectionUpdateData	An interface for updating data on the `SectionCollection` object, for use in `sectionCollection.set({ ... })`.
Word.Interfaces.SectionData	An interface describing the data returned by calling `section.toJSON()`.
Word.Interfaces.SectionLoadOptions	Represents a section in a Word document.
Word.Interfaces.SectionUpdateData	An interface for updating data on the `Section` object, for use in `section.set({ ... })`.

Word.Interfaces.SettingCollectionData	An interface describing the data returned by calling `settingCollection.toJSON()`.
Word.Interfaces.SettingCollectionLoadOptions	Contains the collection of Word.Setting objects.
Word.Interfaces.SettingCollectionUpdateData	An interface for updating data on the `SettingCollection` object, for use in `settingCollection.set({ ... })`.
Word.Interfaces.SettingData	An interface describing the data returned by calling `setting.toJSON()`.
Word.Interfaces.SettingLoadOptions	Represents a setting of the add-in.
Word.Interfaces.SettingUpdateData	An interface for updating data on the `Setting` object, for use in `setting.set({ ... })`.
Word.Interfaces.ShadingData	An interface describing the data returned by calling `shading.toJSON()`.
Word.Interfaces.ShadingLoadOptions	Represents the shading object.
Word.Interfaces.ShadingUniversalData	An interface describing the data returned by calling `shadingUniversal.toJSON()`.
Word.Interfaces.ShadingUniversalLoadOptions	Represents the `ShadingUniversal` object, which manages shading for a range, paragraph, frame, or table.
Word.Interfaces.ShadingUniversalUpdateData	An interface for updating data on the `ShadingUniversal` object, for use in `shadingUniversal.set({ ... })`.
Word.Interfaces.ShadingUpdateData	An interface for updating data on the `Shading` object, for use in `shading.set({ ... })`.
Word.Interfaces.ShadowFormatData	An interface describing the data returned by calling `shadowFormat.toJSON()`.
Word.Interfaces.ShadowFormat	Represents the shadow formatting for a shape or text in Word.

LoadOptions	
Word.Interfaces.ShadowFormatUpdateData	An interface for updating data on the `ShadowFormat` object, for use in `shadowFormat.set({ ... })`.
Word.Interfaces.ShapeCollectionData	An interface describing the data returned by calling `shapeCollection.toJSON()`.
Word.Interfaces.ShapeCollectionLoadOptions	Contains a collection of Word.Shape objects. Currently, only the following shapes are supported: text boxes, geometric shapes, groups, pictures, and canvases.
Word.Interfaces.ShapeCollectionUpdateData	An interface for updating data on the `ShapeCollection` object, for use in `shapeCollection.set({ ... })`.
Word.Interfaces.ShapeData	An interface describing the data returned by calling `shape.toJSON()`.
Word.Interfaces.ShapeFillData	An interface describing the data returned by calling `shapeFill.toJSON()`.
Word.Interfaces.ShapeFillLoadOptions	Represents the fill formatting of a shape object.
Word.Interfaces.ShapeFillUpdateData	An interface for updating data on the `ShapeFill` object, for use in `shapeFill.set({ ... })`.
Word.Interfaces.ShapeGroupData	An interface describing the data returned by calling `shapeGroup.toJSON()`.
Word.Interfaces.ShapeGroupLoadOptions	Represents a shape group in the document. To get the corresponding Shape object, use ShapeGroup.shape.
Word.Interfaces.ShapeGroupUpdateData	An interface for updating data on the `ShapeGroup` object, for use in `shapeGroup.set({ ... })`.
Word.Interfaces.ShapeLoadOptions	Represents a shape in the header, footer, or document body. Currently, only the following shapes are supported: text boxes, geometric shapes, groups, pictures, and canvases.
Word.Interfaces.ShapeTextWrapData	An interface describing the data returned by calling `shapeTextWrap.toJSON()`.

Word.Interfaces.ShapeTextWrapLoadOptions	Represents all the properties for wrapping text around a shape.
Word.Interfaces.ShapeTextWrapUpdateData	An interface for updating data on the `ShapeTextWrap` object, for use in `shapeTextWrap.set({ ... })`.
Word.Interfaces.ShapeUpdateData	An interface for updating data on the `Shape` object, for use in `shape.set({ ... })`.
Word.Interfaces.SourceCollectionData	An interface describing the data returned by calling `sourceCollection.toJSON()`.
Word.Interfaces.SourceCollectionLoadOptions	Represents a collection of Word.Source objects.
Word.Interfaces.SourceCollectionUpdateData	An interface for updating data on the `SourceCollection` object, for use in `sourceCollection.set({ ... })`.
Word.Interfaces.SourceData	An interface describing the data returned by calling `source.toJSON()`.
Word.Interfaces.SourceLoadOptions	Represents an individual source, such as a book, journal article, or interview.
Word.Interfaces.StyleCollectionData	An interface describing the data returned by calling `styleCollection.toJSON()`.
Word.Interfaces.StyleCollectionLoadOptions	Contains a collection of Word.Style objects.
Word.Interfaces.StyleCollectionUpdateData	An interface for updating data on the `StyleCollection` object, for use in `styleCollection.set({ ... })`.
Word.Interfaces.StyleData	An interface describing the data returned by calling `style.toJSON()`.
Word.Interfaces.StyleLoadOptions	Represents a style in a Word document.
Word.Interfaces.StyleUpdateData	An interface for updating data on the `Style` object, for use in `style.set({ ... })`.

Word.Interfaces.TableBorderData	An interface describing the data returned by calling `tableBorder.toJSON()`.
Word.Interfaces.TableBorderLoadOptions	Specifies the border style.
Word.Interfaces.TableBorderUpdateData	An interface for updating data on the `TableBorder` object, for use in `tableBorder.set({ ... })`.
Word.Interfaces.TableCellCollectionData	An interface describing the data returned by calling `tableCellCollection.toJSON()`.
Word.Interfaces.TableCellCollectionLoadOptions	Contains the collection of the document's TableCell objects.
Word.Interfaces.TableCellCollectionUpdateData	An interface for updating data on the `TableCellCollection` object, for use in `tableCellCollection.set({ ... })`.
Word.Interfaces.TableCellData	An interface describing the data returned by calling `tableCell.toJSON()`.
Word.Interfaces.TableCellLoadOptions	Represents a table cell in a Word document.
Word.Interfaces.TableCellUpdateData	An interface for updating data on the `TableCell` object, for use in `tableCell.set({ ... })`.
Word.Interfaces.TableCollectionData	An interface describing the data returned by calling `tableCollection.toJSON()`.
Word.Interfaces.TableCollectionLoadOptions	Contains the collection of the document's Table objects.
Word.Interfaces.TableCollectionUpdateData	An interface for updating data on the `TableCollection` object, for use in `tableCollection.set({ ... })`.
Word.Interfaces.TableColumnCollectionData	An interface describing the data returned by calling `tableColumnCollection.toJSON()`.

Word.Interfaces.TableColumnCollectionLoadOptions	Represents a collection of Word.TableColumn objects in a Word document.
Word.Interfaces.TableColumnCollectionUpdateData	An interface for updating data on the `TableColumnCollection` object, for use in `tableColumnCollection.set({ ... })`.
Word.Interfaces.TableColumnData	An interface describing the data returned by calling `tableColumn.toJSON()`.
Word.Interfaces.TableColumnLoadOptions	Represents a table column in a Word document.
Word.Interfaces.TableColumnUpdateData	An interface for updating data on the `TableColumn` object, for use in `tableColumn.set({ ... })`.
Word.Interfaces.TableData	An interface describing the data returned by calling `table.toJSON()`.
Word.Interfaces.TableLoadOptions	Represents a table in a Word document.
Word.Interfaces.TableRowCollectionData	An interface describing the data returned by calling `tableRowCollection.toJSON()`.
Word.Interfaces.TableRowCollectionLoadOptions	Contains the collection of the document's TableRow objects.
Word.Interfaces.TableRowCollectionUpdateData	An interface for updating data on the `TableRowCollection` object, for use in `tableRowCollection.set({ ... })`.
Word.Interfaces.TableRowData	An interface describing the data returned by calling `tableRow.toJSON()`.
Word.Interfaces.TableRowLoadOptions	Represents a row in a Word document.
Word.Interfaces.TableRowUpdateData	An interface for updating data on the `TableRow` object, for use in `tableRow.set({ ... })`.

Word.Interfaces.TableStyleData	An interface describing the data returned by calling `tableStyle.toJSON()`.
Word.Interfaces.TableStyleLoadOptions	Represents the TableStyle object.
Word.Interfaces.TableStyleUpdateData	An interface for updating data on the `TableStyle` object, for use in `tableStyle.set({ ... })`.
Word.Interfaces.TableUpdateData	An interface for updating data on the `Table` object, for use in `table.set({ ... })`.
Word.Interfaces.TabStopCollectionData	An interface describing the data returned by calling `tabStopCollection.toJSON()`.
Word.Interfaces.TabStopCollectionLoadOptions	Represents a collection of tab stops in a Word document.
Word.Interfaces.TabStopCollectionUpdateData	An interface for updating data on the `TabStopCollection` object, for use in `tabStopCollection.set({ ... })`.
Word.Interfaces.TabStopData	An interface describing the data returned by calling `tabStop.toJSON()`.
Word.Interfaces.TabStopLoadOptions	Represents a tab stop in a Word document.
Word.Interfaces.TemplateCollectionData	An interface describing the data returned by calling `templateCollection.toJSON()`.
Word.Interfaces.TemplateCollectionLoadOptions	Contains a collection of Word.Template objects that represent all the templates that are currently available. This collection includes open templates, templates attached to open documents, and global templates loaded in the **Templates and Add-ins** dialog box. To learn how to access this dialog in the Word UI, see Load or unload a template or add-in program.
Word.Interfaces.TemplateCollectionUpdateData	An interface for updating data on the `TemplateCollection` object, for use in `templateCollection.set({ ... })`.
Word.Interfaces.TemplateData	An interface describing the data returned by calling `template.toJSON()`.

Word.Interfaces.TemplateLoadOptions	Represents a document template.
Word.Interfaces.TemplateUpdateData	An interface for updating data on the `Template` object, for use in `template.set({ ... })`.
Word.Interfaces.TextColumnCollectionData	An interface describing the data returned by calling `textColumnCollection.toJSON()`.
Word.Interfaces.TextColumnCollectionLoadOptions	A collection of Word.TextColumn objects that represent all the columns of text in the document or a section of the document.
Word.Interfaces.TextColumnCollectionUpdateData	An interface for updating data on the `TextColumnCollection` object, for use in `textColumnCollection.set({ ... })`.
Word.Interfaces.TextColumnData	An interface describing the data returned by calling `textColumn.toJSON()`.
Word.Interfaces.TextColumnLoadOptions	Represents a single text column in a section.
Word.Interfaces.TextColumnUpdateData	An interface for updating data on the `TextColumn` object, for use in `textColumn.set({ ... })`.
Word.Interfaces.TextFrameData	An interface describing the data returned by calling `textFrame.toJSON()`.
Word.Interfaces.TextFrameLoadOptions	Represents the text frame of a shape object.
Word.Interfaces.TextFrameUpdateData	An interface for updating data on the `TextFrame` object, for use in `textFrame.set({ ... })`.
Word.Interfaces.ThreeDimensionalFormatData	An interface describing the data returned by calling `threeDimensionalFormat.toJSON()`.
Word.Interfaces.ThreeDimensional	Represents a shape's three-dimensional formatting.

Word.Interfaces.ThreeDimensionalFormatLoadOptions	
Word.Interfaces.ThreeDimensionalFormatUpdateData	An interface for updating data on the `ThreeDimensionalFormat` object, for use in `threeDimensionalFormat.set({ ... })`.
Word.Interfaces.TrackedChangeCollectionData	An interface describing the data returned by calling `trackedChangeCollection.toJSON()`.
Word.Interfaces.TrackedChangeCollectionLoadOptions	Contains a collection of Word.TrackedChange objects.
Word.Interfaces.TrackedChangeCollectionUpdateData	An interface for updating data on the `TrackedChangeCollection` object, for use in `trackedChangeCollection.set({ ... })`.
Word.Interfaces.TrackedChangeData	An interface describing the data returned by calling `trackedChange.toJSON()`.
Word.Interfaces.TrackedChangeLoadOptions	Represents a tracked change in a Word document.
Word.Interfaces.ViewData	An interface describing the data returned by calling `view.toJSON()`.
Word.Interfaces.ViewLoadOptions	Contains the view attributes (such as show all, field shading, and table gridlines) for a window or pane.
Word.Interfaces.ViewUpdateData	An interface for updating data on the `View` object, for use in `view.set({ ... })`.
Word.Interfaces.WindowCollectionData	An interface describing the data returned by calling `windowCollection.toJSON()`.
Word.Interfaces.WindowCollectionLoadOptions	Represents the collection of window objects.
Word.Interfaces.WindowCollectionUpdateData	An interface for updating data on the `WindowCollection` object, for use in `windowCollection.set({ ... })`.

Word.Interfaces.WindowData	An interface describing the data returned by calling `window.toJSON()`.
Word.Interfaces.WindowLoadOptions	Represents the window that displays the document. A window can be split to contain multiple reading panes.
Word.Interfaces.WindowUpdateData	An interface for updating data on the `Window` object, for use in `window.set({ ... })`.
Word.Interfaces.XmlMappingData	An interface describing the data returned by calling `xmlMapping.toJSON()`.
Word.Interfaces.XmlMappingLoadOptions	Represents the XML mapping on a Word.ContentControl object between custom XML and that content control. An XML mapping is a link between the text in a content control and an XML element in the custom XML data store for this document.
Word.Interfaces.XmlMappingUpdateData	An interface for updating data on the `XmlMapping` object, for use in `xmlMapping.set({ ... })`.
Word.ListFormatCountNumberedItemsOptions	Represents options for counting numbered items in a range.
Word.ListTemplateApplyOptions	Represents options for applying a list template to a range.
Word.ParagraphAddedEventArgs	Provides information about the paragraphs that raised the paragraphAdded event.
Word.ParagraphChangedEventArgs	Provides information about the paragraphs that raised the paragraphChanged event.
Word.ParagraphDeletedEventArgs	Provides information about the paragraphs that raised the paragraphDeleted event.
Word.TabStopAddOptions	Specifies the options for adding to a Word.TabStopCollection object.
Word.TextColumnAddOptions	Represents options for a new text column in a document or section of a document.
Word.WindowCloseOptions	The options that define whether to save changes before closing and whether to route the document.
Word.Window	The options for scrolling through the specified pane or window page by page.

PageScrollOptions	
Word.Window ScrollOptions	The options that scrolls a window or pane by the specified number of units defined by the calling method.
Word.XmlSet MappingOptions	The options that define the prefix mapping and the source of the custom XML data.

Enums

Expand table

Word.Alignment	
Word.Annotation State	Represents the state of the annotation.
Word.Arrowhead Length	Specifies the length of the arrowhead at the end of a line.
Word.Arrowhead Style	Specifies the style of the arrowhead at the end of a line.
Word.Arrowhead Width	Specifies the width of the arrowhead at the end of a line.
Word.Baseline Alignment	Represents the type of baseline alignment.
Word.BevelType	Indicates the bevel type of a Word.ThreeDimensionalFormat object.
Word.BodyType	Represents the types of body objects.
Word.BorderLine Style	Specifies the border style for an object.
Word.BorderLocation	
Word.BorderType	
Word.BorderWidth	Represents the width of a style's border.
Word.BreakType	Specifies the form of a break.
Word.BuildingBlock Type	Specifies the type of building block.
Word.BuiltInStyle Name	Represents the built-in style in a Word document.

Word.CalendarType	Calendar types.
Word.CellPaddingLocation	
Word.ChangeTrackingMode	Represents the possible change tracking modes.
Word.ChangeTrackingState	Specify the track state when ChangeTracking is on.
Word.ChangeTrackingVersion	Specify the current version or the original version of the text.
Word.CharacterCase	Specifies the case of the text in the specified range.
Word.CharacterWidth	Specifies the character width of the text in the specified range.
Word.CloseBehavior	Specifies the close behavior for `Document.close`.
Word.ColorIndex	Represents color index values in a Word document.
Word.ColorType	Specifies the color type.
Word.ColumnWidth	Specifies the column width options in a Word document.
Word.CommentChangeType	Represents how the comments in the event were changed.
Word.CompareTarget	Specifies the target document for displaying document comparison differences.
Word.ContentControlAppearance	ContentControl appearance.
Word.ContentControlDateStorageFormat	Date storage formats for Word.DatePickerContentControl.
Word.ContentControlLevel	Content control level types.
Word.ContentControlState	Represents the state of the content control.
Word.ContentControlType	Specifies supported content control types and subtypes.
Word.Continue	Specifies whether the formatting from the previous list can be continued.
Word.CritiqueColorScheme	Represents the color scheme of a critique in the document, affecting underline and highlight.

Word.CustomXmlNodeType	Represents the type of a Word.CustomXmlNode.
Word.CustomXmlValidationErrorType	Represents the type of a Word.CustomXmlValidationError.
Word.DefaultListBehavior	Specifies the default list behavior for a list.
Word.DocPartInsertType	Specifies how a building block is inserted into a document.
Word.DocumentPropertyType	
Word.DropPosition	Represents the position of a dropped capital letter.
Word.EmphasisMark	Specifies the type of emphasis mark to use for a character or designated character string.
Word.ErrorCodes	
Word.EventSource	An enum that specifies an event's source. It can be local or remote (through coauthoring).
Word.EventType	Provides information about the type of a raised event.
Word.ExtrusionColorType	Specifies whether the extrusion color is based on the extruded shape's fill (the front face of the extrusion) and automatically changes when the shape's fill changes, or whether the extrusion color is independent of the shape's fill.
Word.FarEastLineBreakLanguageId	Represents the East Asian language to use when breaking lines of text in the specified document or template.
Word.FarEastLineBreakLevel	Represents the level of line breaking to use for East Asian languages in the specified document or template.
Word.FieldKind	Represents the kind of field. Indicates how the field works in relation to updating.
Word.FieldShading	Specifies the field shading options in a Word document.
Word.FieldType	Represents the type of Field.
Word.FillType	Specifies a shape's fill type.
Word.FlowDirection	Specifies the direction in which text flows from one text column to the next.
Word.FrameSizeRule	Represents how Word interprets the rule used to determine the height or width of a Word.Frame.
Word.GeometricShapeType	Specifies the shape type for a `GeometricShape` object.

Word.GradientColorType	Specifies the type of gradient used in a shape's fill.
Word.GradientStyle	Specifies the style for a gradient fill.
Word.GutterPosition	Specifies where the gutter appears in the document.
Word.GutterStyle	Specifies whether the gutter style should conform to left-to-right text flow or right-to-left text flow.
Word.HeaderFooterType	
Word.HeadingSeparator	Specifies the type of separator to use for headings.
Word.HorizontalInVerticalType	Specifies the format for horizontal text set within vertical text.
Word.HyperlinkType	Specifies the hyperlink type.
Word.ImageFormat	
Word.ImeMode	Specifies the IME (Input Method Editor) mode.
Word.ImportedStylesConflictBehavior	Specifies how to handle any conflicts, that is, when imported styles have the same name as existing styles in the current document.
Word.IndexFilter	Specifies the filter type for an index.
Word.IndexFormat	Specifies the format for an index.
Word.IndexSortBy	Specifies how an index is sorted.
Word.IndexType	Specifies the type of index to create.
Word.InsertLocation	The insertion location types.
Word.JustificationMode	Specifies the character spacing adjustment for a document.
Word.Kana	Specifies the Kana type.
Word.LanguageId	Represents the language ID of a Word document.
Word.LayoutMode	Specifies how text is laid out in the layout mode for the current document.
Word.Ligature	Specifies the type of ligature applied to a font.
Word.LightRigType	Indicates the effects lighting for an object.
Word.LineDashStyle	Specifies the dash style for a line.

Word.LineFormatStyle	Specifies the style for a line.
Word.LineSpacing	Represents the type of line spacing.
Word.LineWidth	Specifies the width of an object's border.
Word.LinkType	Specifies the type of link.
Word.ListApplyTo	Specifies the portion of a list to which to apply a list template.
Word.ListBuiltInNumberStyle	
Word.ListBullet	
Word.ListLevelType	
Word.ListNumbering	
Word.ListType	Represents the list type.
Word.LocationRelation	
Word.NoteItemType	Note item type
Word.NumberForm	Specifies the number form setting for an OpenType font.
Word.NumberingRule	Specifies the numbering rule to apply.
Word.NumberSpacing	Specifies the number spacing setting for an OpenType font.
Word.NumberType	Specifies the type of numbers in a list.
Word.OleVerb	Specifies the action associated with the verb that the OLE object should perform.
Word.OutlineLevel	Represents the outline levels.
Word.PageBorderArt	Specifies the graphical page border setting of a page.
Word.PageColor	Specifies the page color options in a Word document.
Word.PageMovementType	Specifies the page movement type in a Word document.
Word.PageOrientation	Specifies a page layout orientation.
Word.PageSetupVerticalAlignment	Specifies the type of vertical alignment to apply.
Word.PaperSize	Specifies a paper size.

Word.PatternType	Specifies the fill pattern used in a shape.
Word.Preferred WidthType	Specifies the preferred unit of measure to use when measuring the width of an item.
Word.PresetCamera	Indicates the effects camera type used by the specified object.
Word.Preset ExtrusionDirection	Specifies the direction that the extrusion's sweep path takes away from the extruded shape (the front face of the extrusion).
Word.Preset GradientType	Specifies which predefined gradient to use to fill a shape.
Word.PresetLighting Direction	Specifies the location of lighting on an extruded (three-dimensional) shape relative to the shape.
Word.PresetLighting Softness	Specifies the intensity of light used on a shape.
Word.PresetMaterial	Specifies the extrusion surface material.
Word.PresetTexture	Specifies texture to be used to fill a shape.
Word.PresetThree DimensionalFormat	Specifies an extrusion (three-dimensional) format.
Word.Range Location	Represents the location of a range. You can get range by calling getRange on different objects such as Word.Paragraph and Word.ContentControl.
Word.Reading LayoutMargin	Specifies the margin options in reading layout view in a Word document.
Word.ReadingOrder	Represents the reading order of text.
Word.ReflectionType	Specifies the type of the Word.ReflectionFormat object.
Word.Relative HorizontalPosition	Represents what the horizontal position of a shape is relative to. For more information about margins, see Change the margins in your Word document.
Word.RelativeSize	Represents what the horizontal or vertical size of a shape is relative to. For more information about margins, see Change the margins in your Word document.
Word.Relative VerticalPosition	Represents what the vertical position of a shape is relative to. For more information about margins, see Change the margins in your Word document.
Word.Revisions BalloonMargin	Specifies the margin for revision balloons in a Word document.
Word.Revisions BalloonWidthType	Specifies the width type for revision balloons in a Word document.
Word.Revisions	Specifies the extent of markup visible in the document.

Markup	
Word.RevisionsMode	Specifies the display mode for tracked changes in a Word document.
Word.RevisionsView	Specifies whether Word displays the original version of a document or a version with revisions and formatting changes applied.
Word.RevisionType	Specifies the revision type.
Word.RulerStyle	Specifies the way Word adjusts the table when the left indent is changed.
Word.SaveBehavior	Specifies the save behavior for `Document.save`.
Word.SaveConfiguration	Specifies the save options.
Word.SectionDirection	Specifies how Word displays the reading order and alignment for the specified sections.
Word.SectionStart	Specifies the type of section break for the specified item.
Word.SeekView	Specifies the seek view options in a Word document.
Word.SelectionMode	This enum sets where the cursor (insertion point) in the document is after a selection.
Word.ShadingTextureType	Represents the shading texture. To learn more about how to apply backgrounds like textures, see Add, change, or delete the background color in Word.
Word.ShadowStyle	Specifies the type of shadowing effect.
Word.ShadowType	Specifies the type of shadow displayed with a shape.
Word.ShapeAutoSize	Determines the type of automatic sizing allowed.
Word.ShapeFillType	Specifies a shape's fill type.
Word.ShapeScaleFrom	Specifies which part of the shape retains its position when the shape is scaled.
Word.ShapeScaleType	Specifies the scale size type of a shape.
Word.ShapeTextOrientation	Specifies the orientation for the text frame in a shape.
Word.ShapeTextVerticalAlignment	Specifies the vertical alignment for the text frame in a shape.
Word.ShapeTextWrapSide	Specifies whether the document text should wrap on both sides of the specified shape, on either the left or right side only, or on the side of the shape that's

	farther from the respective page margin.
Word.ShapeTextWrapType	Specifies how to wrap document text around a shape. For more details, see the "Text Wrapping" tab of Layout options .
Word.ShapeType	Represents the shape type.
Word.ShowSourceDocuments	Specifies the source documents to show.
Word.SpecialPane	Specifies the special pane options in a Word document.
Word.StoryType	Specifies the type of story in a Word document.
Word.StyleType	Represents the type of style.
Word.StylisticSet	Specifies the stylistic set to apply to the font.
Word.TabAlignment	Represents the alignment of a tab stop.
Word.TabLeader	Specifies the tab leader style.
Word.TemplateType	Specifies the type of template.
Word.TextboxTightWrap	Represents the type of tight wrap for a text box.
Word.TextureAlignment	Specifies the alignment (the origin of the coordinate grid) for the tiling of the texture fill.
Word.TextureType	Specifies the texture type for the selected fill.
Word.ThemeColorIndex	Specifies the theme colors for document themes.
Word.TrackedChangeType	TrackedChange type.
Word.TrailingCharacter	Represents the character inserted after the list item mark.
Word.TwoLinesInOneType	Specifies the two lines in one type.
Word.Underline	Specifies the underline type.
Word.UnderlineType	The supported styles for underline format.
Word.VerticalAlignment	
Word.ViewType	Specifies the view type in a Word document.
Word.WindowState	Represents the state of the window.

Word.WindowType	Specifies the type of the window.

Functions

Expand table

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

Function Details

Word.run(objects, batch)

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

```TypeScript
export function run<T>(objects: OfficeExtension.ClientObject[], batch:
(context: Word.RequestContext) => Promise<T>): Promise<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: Word.RequestContext) => Promise<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 Word application. Since the Office add-in and the Word application run in two different processes, the RequestContext is required to get access to the Word object model from the add-in.

Returns

Promise<T>

Word.run(object, batch)

Executes a batch script that performs actions on the Word 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:
Word.RequestContext) => Promise<T>): Promise<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: Word.RequestContext) => Promise<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 Word application. Since the Office add-in and the Word application run in two different processes, the RequestContext is required to get access to the Word object model from the add-in.

Returns

Promise<T>

Word.run(batch)

Executes a batch script that performs actions on the Word 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: Word.RequestContext) => Promise<T>):
Promise<T>;
```

Parameters

`batch` (context: Word.RequestContext) => Promise<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 Word application. Since the Office add-in and the Word application run in two different processes, the RequestContext is required to get access to the Word object model from the add-in.

Returns

Promise<T>

Word JavaScript object model in Office Add-ins

Article • 05/30/2025

This article describes concepts that are fundamental to using the Word JavaScript API to build add-ins.

Office.js APIs for Word

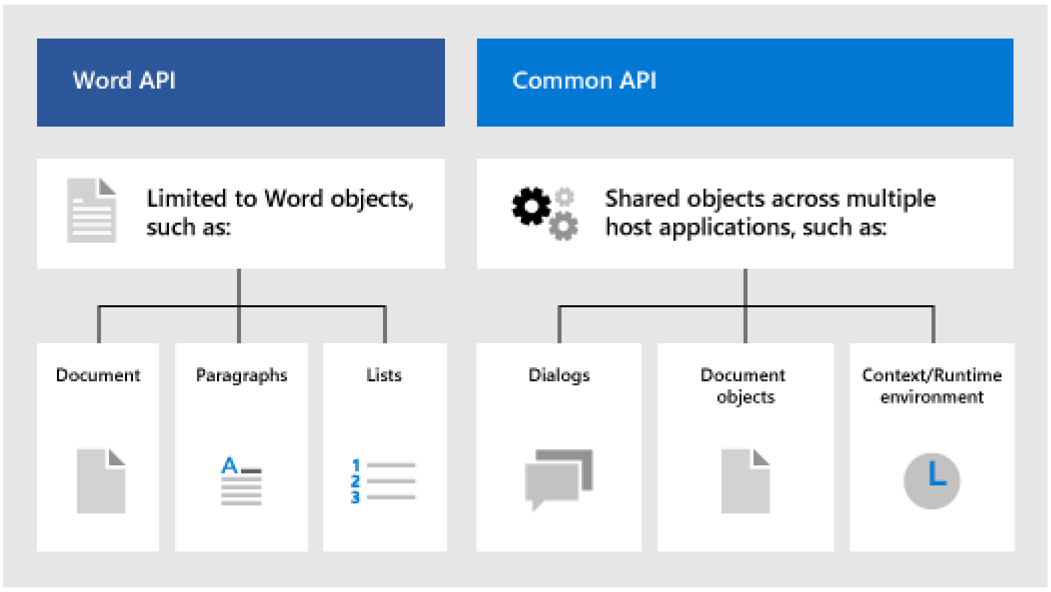
A Word add-in interacts with objects in Word by using the Office JavaScript API. This includes two JavaScript object models:

- **Word JavaScript API**: The Word JavaScript API provides strongly-typed objects that work with the document, ranges, tables, lists, formatting, and more. To learn about the asynchronous nature of the Word APIs and how they work with the document, 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 Word JavaScript API to develop the majority of functionality in add-ins that target Word, you'll also use objects in the Common API. For example:

- Office.Context: The `Context` object represents the runtime environment of the add-in and provides access to key objects of the API. It consists of document 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 Word application where the add-in is running.
- Office.Document: The `Office.Document` object provides the `getFileAsync()` method, which you can use to download the Word file where the add-in is running. This is separate from the Word.Document object.


Word API
Common API
Limited to Word objects, such as:
Shared objects across multiple host applications, such as:
Document
Paragraphs
Lists
Dialogs
Document objects
Context/Runtime environment

Word-specific object model

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

- The document contains sections, pages, and document-level entities such as settings and custom XML parts.
- A section contains a body.
- A body has paragraphs, content controls, and range objects, among others.
- A range is a contiguous area of content, including text, whitespace, tables, and images. The Word.Range object contains most of the text manipulation methods.
- A list contains numbered or bulleted paragraphs.
- The document is contained in a window.
- A window has panes. A pane surrounds the visible area of the document.

For the full set of objects supported by the Word JavaScript API, see Word JavaScript API.

See also

- Word JavaScript API overview
- Build your first Word add-in
- Word add-in tutorial
- Word JavaScript API reference
- Learn about the Microsoft 365 Developer Program

Create a dictionary task pane add-in

Article • 02/12/2025

This article shows you an example of a task pane add-in with an accompanying web service that provides dictionary definitions or thesaurus synonyms for the user's current selection in a Word document.

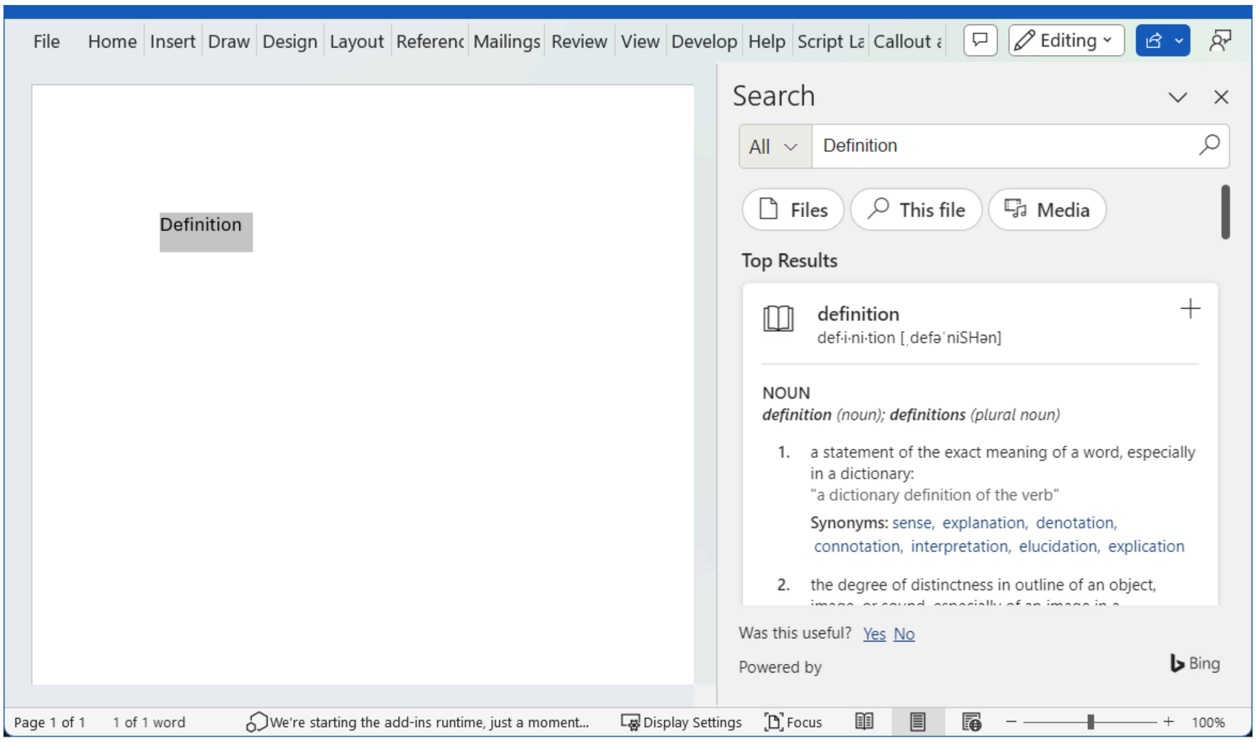
A dictionary Office Add-in is based on the standard task pane add-in with additional features to support querying and displaying definitions from a dictionary XML web service in additional places in the Office application's UI.

In a typical dictionary task pane add-in, a user selects a word or phrase in their document, and the JavaScript logic behind the add-in passes this selection to the dictionary provider's XML web service. The dictionary provider's webpage then updates to show the definitions for the selection to the user.

The XML web service component returns up to three definitions in the format defined by the example OfficeDefinitions XML schema, which are then displayed to the user in other places in the hosting Office application's UI.

Figure 1 shows the selection and display experience for a Bing-branded dictionary add-in that's running in Word.

Figure 1. Dictionary add-in displaying definitions for the selected word



definition
def·i·ni·tion [ˌdefəˈniSHən]

NOUN
definition *(noun);* ***definitions*** *(plural noun)*

1. a statement of the exact meaning of a word, especially in a dictionary:
"a dictionary definition of the verb"
Synonyms: sense, explanation, denotation, connotation, interpretation, elucidation, explication
2. the degree of distinctness in outline of an object,

It's up to you to determine if selecting the **See More** link in the dictionary add-in's HTML UI displays more information within the task pane or opens a separate window to the full webpage for the selected word or phrase.

Figure 2 shows the **Define** command in the context menu that enables users to quickly launch installed dictionaries. Figures 3 through 5 show the places in the Office UI where the dictionary XML services are used to provide definitions in Word.

Figure 2. Define command in the context menu

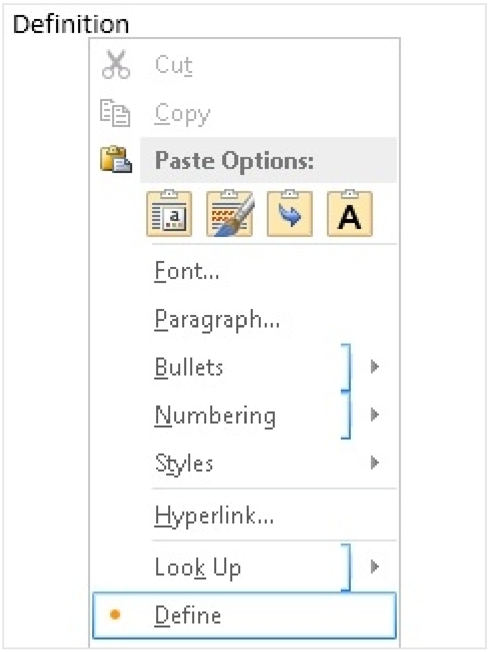
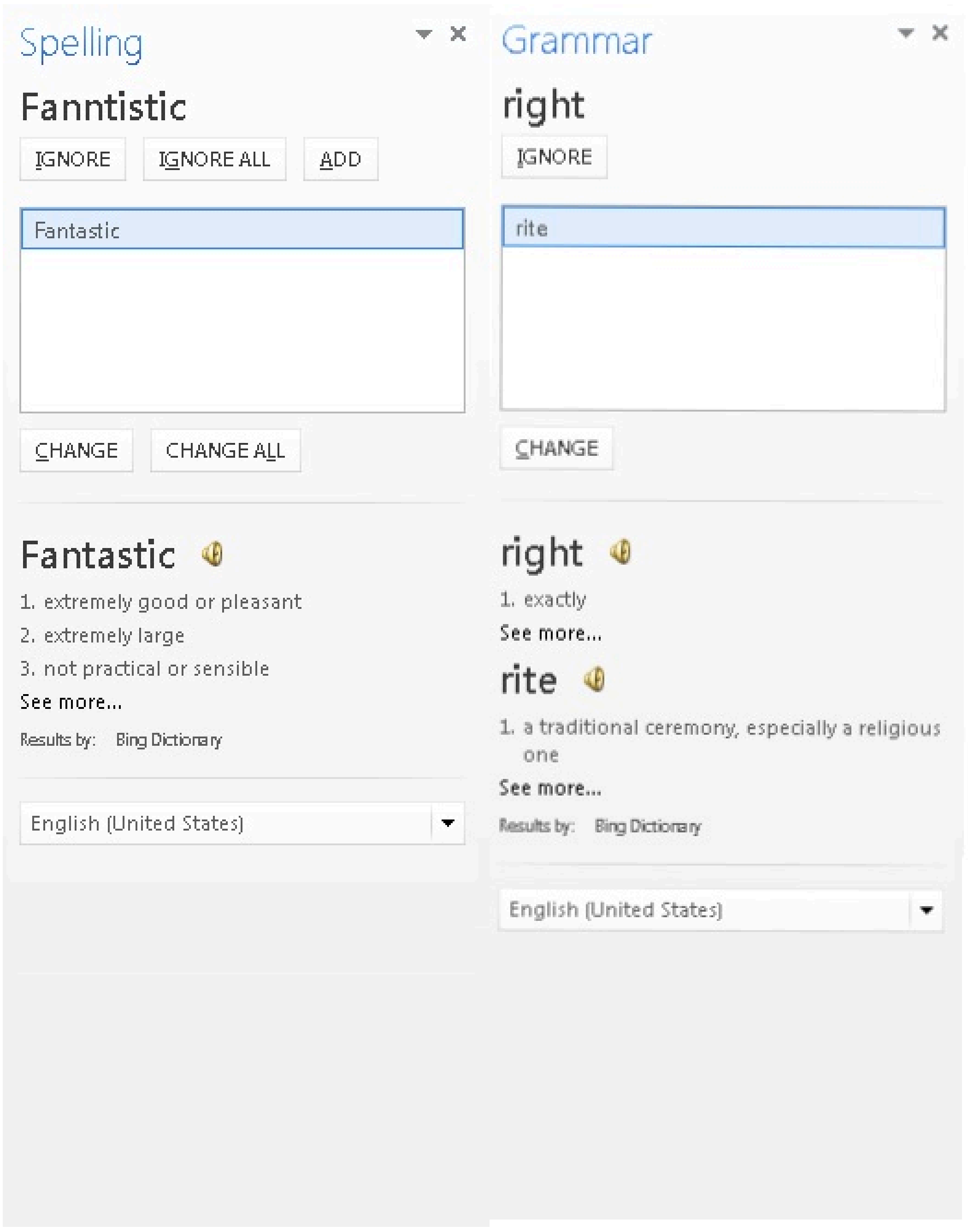


Figure 3. Definitions in the Spelling and Grammar panes


Spelling

Fanntistic

IGNORE IGNORE ALL ADD

Fantastic

CHANGE CHANGE ALL

Fantastic

1. extremely good or pleasant
2. extremely large
3. not practical or sensible

See more...

Results by: Bing Dictionary

English (United States)

Grammar

right

IGNORE

rite

CHANGE

right

1. exactly

See more...

rite

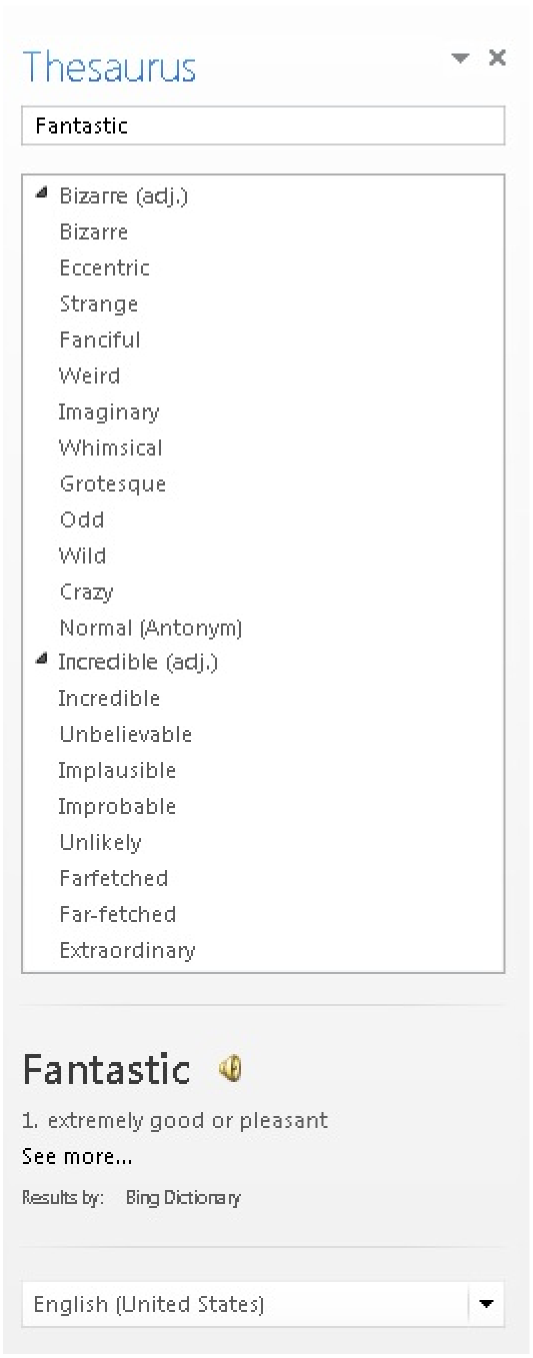
1. a traditional ceremony, especially a religious one

See more...

Results by: Bing Dictionary

English (United States)

Figure 4. Definitions in the Thesaurus pane



Thesaurus

Fantastic

- Bizarre (adj.)
  - Bizarre
  - Eccentric
  - Strange
  - Fanciful
  - Weird
  - Imaginary
  - Whimsical
  - Grotesque
  - Odd
  - Wild
  - Crazy
  - Normal (Antonym)
- Incredible (adj.)
  - Incredible
  - Unbelievable
  - Implausible
  - Improbable
  - Unlikely
  - Farfetched
  - Far-fetched
  - Extraordinary

Fantastic

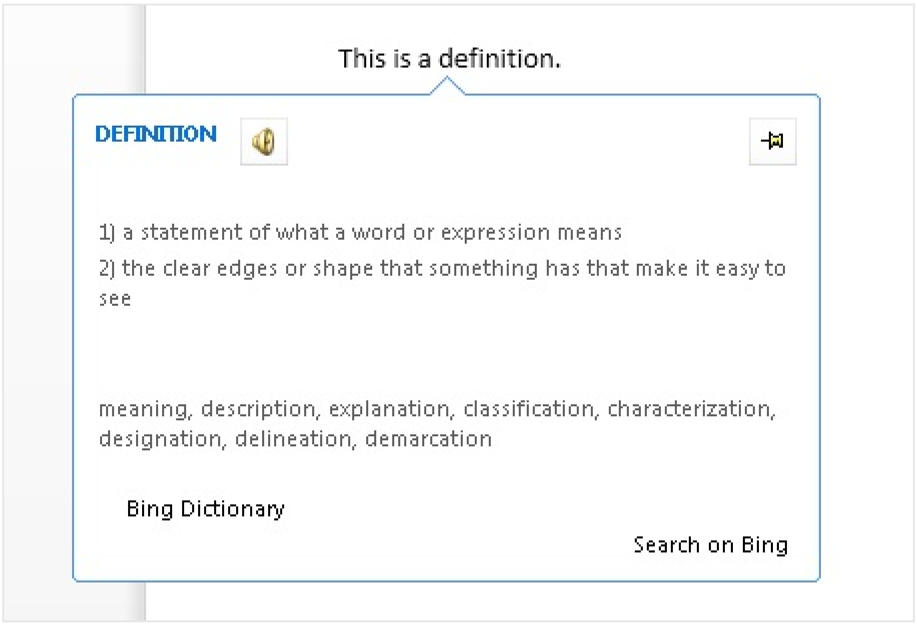
1. extremely good or pleasant

See more...

Results by: Bing Dictionary

English (United States)

Figure 5. Definitions in Reading Mode


This is a definition.

DEFINITION

1) a statement of what a word or expression means
2) the clear edges or shape that something has that make it easy to see

meaning, description, explanation, classification, characterization, designation, delineation, demarcation

Bing Dictionary

Search on Bing

To create a task pane add-in that provides a dictionary lookup, create two main components.

- An XML web service that looks up definitions from a dictionary service, and then returns those values in an XML format that can be consumed and displayed by the dictionary add-in.
- A task pane add-in that submits the user's current selection to the dictionary web service, displays definitions, and can optionally insert those values into the document.

The following sections provide examples of how to create these components.

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).

Next, create a Word add-in project in Visual Studio.

1. In Visual Studio, choose **Create a new project**.

2. Using the search box, enter **add-in**. Choose **Word Web Add-in**, then select **Next**.

3. Name your project and select **Create**.

4. Visual Studio creates a solution and its two projects appear in **Solution Explorer**. The **Home.html** file opens in Visual Studio.

To learn more about the projects in a Word add-in solution, see the quick start.

Create a dictionary XML web service

The XML web service must return queries to the web service as XML that conforms to the OfficeDefinitions XML schema. The following two sections describe the OfficeDefinitions XML schema, and provide an example of how to code an XML web service that returns queries in that XML format.

OfficeDefinitions XML schema

The following code shows sample XSD for the OfficeDefinitions XML schema example.

XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xs="https://www.w3.org/2001/XMLSchema"
  targetNamespace="http://schemas.microsoft.com/contoso/OfficeDefinitions"
  xmlns="http://schemas.microsoft.com/contoso/OfficeDefinitions">
  <xs:element name="Result">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="SeeMoreURL" type="xs:anyURI"/>
        <xs:element name="Definitions" type="DefinitionListType"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="DefinitionListType">
    <xs:sequence>
      <xs:element name="Definition" maxOccurs="3">
        <xs:simpleType>
          <xs:restriction base="xs:normalizedString">
            <xs:maxLength value="400"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

Returned XML consists of a root **<Result>** element that contains a **<Definitions>** element with zero to three **<Definition>** child elements. Each child element contains definitions that are at most 400 characters in length. Additionally, the URL to the full page on the dictionary site must be provided in the **<SeeMoreURL>** element. The following example shows the structure of returned XML that conforms to the OfficeDefinitions schema.

```XML
<?xml version="1.0" encoding="utf-8"?>
<Result xmlns="http://schemas.microsoft.com/contoso/OfficeDefinitions">
  <SeeMoreURL xmlns="">https://www.bing.com/search?q=example</SeeMoreURL>
  <Definitions xmlns="">
    <Definition>Definition1</Definition>
    <Definition>Definition2</Definition>
    <Definition>Definition3</Definition>
  </Definitions>
 </Result>
```

Sample dictionary XML web service

The following C# code provides an example of how to write code for an XML web service that returns the result of a dictionary query in the OfficeDefinitions XML format.

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Xml;
using System.Text;
using System.IO;
using System.Net;
using System.Web.Script.Services;

/// <summary>
/// Summary description for _Default.
/// </summary>
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
// To allow this web service to be called from script, using ASP.NET AJAX,
include the following line.
[ScriptService]
public class WebService : System.Web.Services.WebService {

    public WebService () {
        // Uncomment the following line if using designed components.
        // InitializeComponent();
    }

    // You can replace this method entirely with your own method that gets
definitions
    // from your data source and then formats it into the example
OfficeDefinitions XML format.
    // If you need a reference for constructing the returned XML, you can
use this example as a basis.
    [WebMethod]
    public XmlDocument Define(string word)
    {

        StringBuilder sb = new StringBuilder();
        XmlWriter writer = XmlWriter.Create(sb);
        {
            writer.WriteStartDocument();

                writer.WriteStartElement("Result",
"http://schemas.microsoft.com/contoso/OfficeDefinitions");

                    // See More URL should be changed to the dictionary
publisher's page for that word on
                    // their website.
                    writer.WriteElementString("SeeMoreURL",
"https://www.bing.com/search?q=" + word);

                    writer.WriteStartElement("Definitions");

                        writer.WriteElementString("Definition", "Definition
1 of " + word);
                        writer.WriteElementString("Definition", "Definition
2 of " + word);
                        writer.WriteElementString("Definition", "Definition
3 of " + word);

                    writer.WriteEndElement(); // End of Definitions element.

                writer.WriteEndElement(); // End of Result element.

            writer.WriteEndDocument();
        }
        writer.Close();

        XmlDocument doc = new XmlDocument();
        doc.LoadXml(sb.ToString());

        return doc;
    }
}
```

To get started with development, you can do the following.

Create the web service

1. Add a **Web Service (ASMX)** to the add-in's web application project in Visual Studio and name it **DictionaryWebService**.
2. Replace the entire content of the associated .asmx.cs file with the preceding C# code sample.

Update the web service markup

1. In the **Solution Explorer**, select the **DictionaryWebService.asmx** file then open its context menu and choose **View Markup**.

2. Replace the contents of DictionaryWebService.asmx with the following code.

   XML

   ```xml
   <%@ WebService Language="C#" CodeBehind="DictionaryWebService.asmx.cs"
   Class="WebService" %>
   ```

Update the web.config

1. In the **Web.config** of the add-in's web application project, add the following to the **<system.web>** node.

   XML

   ```xml
   <webServices>
     <protocols>
       <add name="HttpGet" />
       <add name="HttpPost" />
     </protocols>
   </webServices>
   ```

2. Save your changes.

Components of a dictionary add-in

A dictionary add-in consists of three main component files:

- An XML-formatted add-in only manifest file that describes the add-in.

  ⓘ **Important**

The JSON-formatted unified manifest for Microsoft 365 doesn't currently support dictionary add-ins.

- An HTML file that provides the add-in's UI.
- A JavaScript file that provides logic to get the user's selection from the document, sends the selection as a query to the web service, and then displays returned results in the add-in's UI.

Example of a dictionary add-in's manifest file

The following is an example manifest file for a dictionary add-in.

XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<OfficeApp xmlns="http://schemas.microsoft.com/office/appforoffice/1.0"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:type="TaskPaneApp">
  <Id>7164e750-dc86-49c0-b548-1bac57abdc7c</Id>
  <Version>15.0</Version>
  <ProviderName>Microsoft Office Demo Dictionary</ProviderName>
  <DefaultLocale>en-us</DefaultLocale>
  <!--DisplayName is the name that will appear in the user's list of
applications.-->
  <DisplayName DefaultValue="Microsoft Office Demo Dictionary" />
  <!--Description is a 2-3 sentence description of this dictionary. -->
  <Description DefaultValue="The Microsoft Office Demo Dictionary is an
example built to demonstrate how a
    publisher can create a dictionary that integrates with Office. It
doesn't return real definitions." />
  <!--IconUrl is the URI for the icon that will appear in the user's list of
applications.-->
  <IconUrl
DefaultValue="http://contoso/_layouts/images/general/office_logo.jpg" />
  <SupportUrl DefaultValue="[Insert the URL of a page that provides support
information for the app]" />
  <!--Hosts specifies the kind of Office application your dictionary add-in
will support.
      You shouldn't have to modify this area.-->
  <Hosts>
    <Host Name="Document"/>
  </Hosts>
  <DefaultSettings>
    <!--SourceLocation is the URL for your dictionary.-->
    <SourceLocation
DefaultValue="http://contoso/ExampleDictionary/DictionaryHome.html" />
  </DefaultSettings>
  <!--Permissions is the set of permissions a user will have to give your
dictionary.
      If you need write access, such as to allow a user to replace the
highlighted word with a synonym,
      use ReadWriteDocument. -->
  <Permissions>ReadDocument</Permissions>
  <Dictionary>
    <!--TargetDialects is the set of regional languages your dictionary
contains. For example, if your
        dictionary applies to Spanish (Mexico) and Spanish (Peru), but not
Spanish (Spain), you can specify
        that here. Do not put more than one language (for example, Spanish
and English) here. Publish
        separate languages as separate dictionaries. -->
    <TargetDialects>
      <TargetDialect>EN-AU</TargetDialect>
      <TargetDialect>EN-BZ</TargetDialect>
      <TargetDialect>EN-CA</TargetDialect>
      <TargetDialect>EN-029</TargetDialect>
      <TargetDialect>EN-HK</TargetDialect>
      <TargetDialect>EN-IN</TargetDialect>
      <TargetDialect>EN-ID</TargetDialect>
      <TargetDialect>EN-IE</TargetDialect>
      <TargetDialect>EN-JM</TargetDialect>
      <TargetDialect>EN-MY</TargetDialect>
      <TargetDialect>EN-NZ</TargetDialect>
      <TargetDialect>EN-PH</TargetDialect>
      <TargetDialect>EN-SG</TargetDialect>
      <TargetDialect>EN-ZA</TargetDialect>
      <TargetDialect>EN-TT</TargetDialect>
      <TargetDialect>EN-GB</TargetDialect>
      <TargetDialect>EN-US</TargetDialect>
      <TargetDialect>EN-ZW</TargetDialect>
    </TargetDialects>
    <!--QueryUri is the address of this dictionary's XML web service (which
is used to put definitions in
        additional contexts, such as the spelling checker.)-->
    <QueryUri
DefaultValue="http://contoso/ExampleDictionary/WebService.asmx/Define?
word="/>
    <!--Citation Text, Dictionary Name, and Dictionary Home Page will be
combined to form the citation line
        (for example, this would produce "Examples by: Contoso",
        where "Contoso" is a hyperlink to http://www.contoso.com).-->
    <CitationText DefaultValue="Examples by: " />
    <DictionaryName DefaultValue="Contoso" />
    <DictionaryHomePage DefaultValue="http://www.contoso.com" />
  </Dictionary>
</OfficeApp>
```

The **<Dictionary>** element and its child elements specific to creating a dictionary add-in's manifest file are described in the following sections. For information about the other elements in the manifest file, see Office Add-ins with the add-in only manifest.

Dictionary element

Specifies settings for dictionary add-ins.

Parent element

<OfficeApp>

Child elements

<TargetDialects>, **<QueryUri>**, **<CitationText>**, **<Name>**, **<DictionaryHomePage>**

Remarks

The **<Dictionary>** element and its child elements are added to the manifest of a task pane add-in when you create a dictionary add-in.

TargetDialects element

Specifies the regional languages that this dictionary supports. Required for dictionary add-ins.

Parent element

<Dictionary>

Child element

<TargetDialect>

Remarks

The **<TargetDialects>** element and its child elements specify the set of regional languages your dictionary contains. For example, if your dictionary applies to both Spanish (Mexico) and Spanish (Peru), but not Spanish (Spain), you can specify that in this element. Do not specify more than one language (e.g., Spanish and English) in this manifest. Publish separate languages as separate dictionaries.

Example

XML

```xml
<TargetDialects>
  <TargetDialect>EN-AU</TargetDialect>
  <TargetDialect>EN-BZ</TargetDialect>
  <TargetDialect>EN-CA</TargetDialect>
  <TargetDialect>EN-029</TargetDialect>
  <TargetDialect>EN-HK</TargetDialect>
  <TargetDialect>EN-IN</TargetDialect>
  <TargetDialect>EN-ID</TargetDialect>
```

```
    <TargetDialect>EN-IE</TargetDialect>
    <TargetDialect>EN-JM</TargetDialect>
    <TargetDialect>EN-MY</TargetDialect>
    <TargetDialect>EN-NZ</TargetDialect>
    <TargetDialect>EN-PH</TargetDialect>
    <TargetDialect>EN-SG</TargetDialect>
    <TargetDialect>EN-ZA</TargetDialect>
    <TargetDialect>EN-TT</TargetDialect>
    <TargetDialect>EN-GB</TargetDialect>
    <TargetDialect>EN-US</TargetDialect>
    <TargetDialect>EN-ZW</TargetDialect>
  </TargetDialects>
```

TargetDialect element

Specifies a regional language that this dictionary supports. Required for dictionary add-ins.

Parent element

<TargetDialects>

Remarks

Specify the value for a regional language in the RFC1766 `language` tag format, such as EN-US.

Example

```XML
<TargetDialect>EN-US</TargetDialect>
```

QueryUri element

Specifies the endpoint for the dictionary query service. Required for dictionary add-ins.

Parent element

<Dictionary>

Remarks

This is the URI of the XML web service for the dictionary provider. The properly escaped query will be appended to this URI.

Example

```XML
<QueryUri DefaultValue="http://msranlc-lingo1/proof.aspx?q="/>
```

CitationText element

Specifies the text to use in citations. Required for dictionary add-ins.

Parent element

<Dictionary>

Remarks

This element specifies the beginning of the citation text that will be displayed on a line below the content that is returned from the web service (for example, "Results by: " or "Powered by: ").

For this element, you can specify values for additional locales by using the **<Override>** element. For example, if a user is running the Spanish SKU of Office, but using an English dictionary, this allows the citation line to read "Resultados por: Bing" rather than "Results by: Bing". For more information about how to specify values for additional locales, see Localization.

Example

```XML
<CitationText DefaultValue="Results by: " />
```

DictionaryName element

Specifies the name of this dictionary. Required for dictionary add-ins.

Parent element

<Dictionary>

Remarks

This element specifies the link text in the citation text. Citation text is displayed on a line below the content that is returned from the web service.

For this element, you can specify values for additional locales.

Example

```XML
<DictionaryName DefaultValue="Bing Dictionary" />
```

DictionaryHomePage element

Specifies the URL of the home page for the dictionary. Required for dictionary add-ins.

Parent element

<Dictionary>

Remarks

This element specifies the link URL in the citation text. Citation text is displayed on a line below the content that is returned from the web service.

For this element, you can specify values for additional locales.

Example

```XML
<DictionaryHomePage DefaultValue="https://www.bing.com" />
```

Update your dictionary add-in's manifest file

1. Open the manifest file in the add-in project.

2. Update the value of the **<ProviderName>** element with your name.

3. Replace the value of the **<DisplayName>** element's **<DefaultValue>** attribute with an appropriate name, for example, "Microsoft Office Demo Dictionary".

4. Replace the value of the **<Description>** element's **<DefaultValue>** attribute with an appropriate description, for example, "The Microsoft Office Demo Dictionary is an example built to demonstrate how a publisher could create a dictionary that integrates with Office. It doesn't return real definitions.".

5. Add the following code after the **<Permissions>** node, replacing "contoso" references with your own company name, then save your changes.

XML

```xml
<Dictionary>
  <!--TargetDialects is the set of regional languages your dictionary
contains. For example, if your
      dictionary applies to Spanish (Mexico) and Spanish (Peru), but
not Spanish (Spain), you can
      specify that here. Do not put more than one language (for
example, Spanish and English) here.
      Publish separate languages as separate dictionaries. -->
  <TargetDialects>
    <TargetDialect>EN-AU</TargetDialect>
    <TargetDialect>EN-BZ</TargetDialect>
    <TargetDialect>EN-CA</TargetDialect>
    <TargetDialect>EN-029</TargetDialect>
    <TargetDialect>EN-HK</TargetDialect>
    <TargetDialect>EN-IN</TargetDialect>
    <TargetDialect>EN-ID</TargetDialect>
    <TargetDialect>EN-IE</TargetDialect>
    <TargetDialect>EN-JM</TargetDialect>
    <TargetDialect>EN-MY</TargetDialect>
    <TargetDialect>EN-NZ</TargetDialect>
    <TargetDialect>EN-PH</TargetDialect>
    <TargetDialect>EN-SG</TargetDialect>
    <TargetDialect>EN-ZA</TargetDialect>
    <TargetDialect>EN-TT</TargetDialect>
    <TargetDialect>EN-GB</TargetDialect>
    <TargetDialect>EN-US</TargetDialect>
    <TargetDialect>EN-ZW</TargetDialect>
  </TargetDialects>
  <!--QueryUri is the address of this dictionary's XML web service
(which is used to put definitions in
      additional contexts, such as the spelling checker.)-->
  <QueryUri DefaultValue="~remoteAppUrl/DictionaryWebService.asmx"/>
  <!--Citation Text, Dictionary Name, and Dictionary Home Page will be
combined to form the citation
      line (for example, this would produce "Examples by: Contoso",
where "Contoso" is a hyperlink to
      http://www.contoso.com).-->
  <CitationText DefaultValue="Examples by: " />
  <DictionaryName DefaultValue="Contoso" />
  <DictionaryHomePage DefaultValue="http://www.contoso.com" />
</Dictionary>
```

Create a dictionary add-in's HTML user interface

The following two examples show the HTML and CSS files for the UI of the Demo Dictionary add-in. To view how the UI is displayed in the add-in's task pane, see Figure 6 following the code. To see how the implementation of the JavaScript provides programming logic for this HTML UI, see Write the JavaScript implementation immediately following this section.

In the add-in's web application project in Visual Studio, you can replace the contents of the **./Home.html** file with the following sample HTML.

HTML

```html
<!DOCTYPE html>
<html>

<head>
    <meta http-equiv="X-UA-Compatible" content="IE=Edge" />

    <!--The title will not be shown but is supplied to ensure valid HTML.-->
    <title>Example Dictionary</title>

    <!--Required library includes.-->
    <script type="text/javascript"
src="https://ajax.microsoft.com/ajax/4.0/1/MicrosoftAjax.js"></script>
    <script src="https://appsforoffice.microsoft.com/lib/1/hosted/office.js"
type="text/javascript"></script>

    <!--Optional library includes.-->
    <script type="text/javascript"
src="https://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.5.1.js"></script>

    <!--App-specific CSS and JS.-->
    <link rel="Stylesheet" type="text/css" href="Home.css" />
    <script type="text/javascript" src="Home.js"></script>
</head>

<body>
    <div id="mainContainer">
        <div>INSTRUCTIONS</div>
        <ol>
            <li>Ensure there's text in the document.</li>
            <li>Select text.</li>
        </ol>
        <div id="header">
            <span id="headword"></span>
        </div>
        <div>DEFINITIONS</div>
        <ol id="definitions">
        </ol>
        <div id="SeeMore">
            <a id="SeeMoreLink" target="_blank">See More...</a>
        </div>
        <div id="message"></div>
    </div>
</body>
```

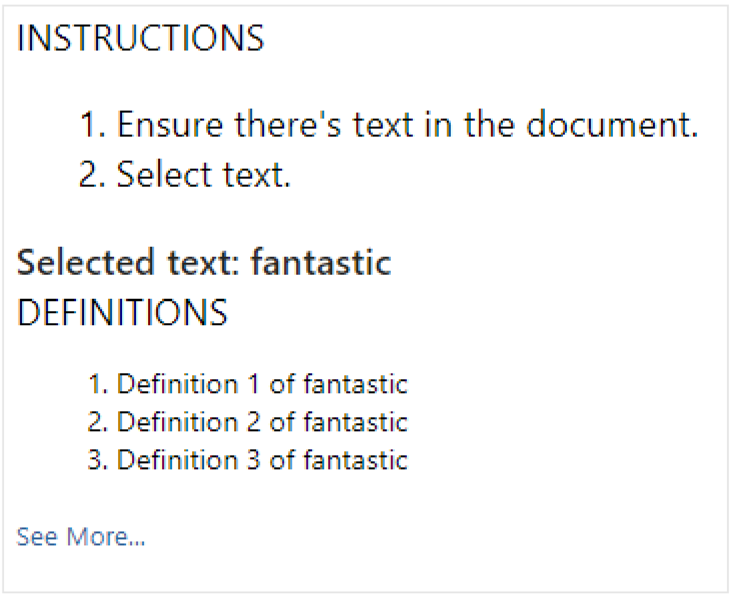
```
</html>
```

The following example shows the contents of the .css file.

In the add-in's web application project in Visual Studio, you can replace the contents of the **./Home.css** file with the following sample CSS.

```css
#mainContainer
{
    font-family: Segoe UI;
    font-size: 11pt;
}

#headword
{
    font-family: Segoe UI Semibold;
    color: #262626;
}

#definitions
{
    font-size: 8.5pt;
}
a
{
    font-size: 8pt;
    color: #336699;
    text-decoration: none;
}
a:visited
{
    color: #993366;
}
a:hover, a:active
{
    text-decoration: underline;
}
```

Figure 6. Demo dictionary UI


INSTRUCTIONS

1. Ensure there's text in the document.
2. Select text.

Selected text: fantastic

DEFINITIONS

1. Definition 1 of fantastic
2. Definition 2 of fantastic
3. Definition 3 of fantastic

See More...

Write the JavaScript implementation

The following example shows the JavaScript implementation in the .js file that's called from the add-in's HTML page to provide the programming logic for the Demo Dictionary add-in. This script uses the XML web service described previously. When placed in the same directory as the example web service, the script will get definitions from that service. It can be used with a public OfficeDefinitions-conforming XML web service by modifying the `xmlServiceURL` variable at the top of the file.

The primary members of the Office JavaScript API (Office.js) that are called from this implementation are shown in the following list.

- The initialize event of the `Office` object, which is raised when the add-in context is initialized, and provides access to a Document object instance that represents the document the add-in is interacting with.
- The addHandlerAsync method of the `Document` object, which is called in the `initialize` function to add an event handler for the SelectionChanged event of the document to listen for user selection changes.
- The getSelectedDataAsync method of the `Document` object, which is called in the `tryUpdatingSelectedWord()` function when the `SelectionChanged` event handler is raised to get the word or phrase the user selected, coerce it to plain text, and then execute the `selectedTextCallback` asynchronous callback function.
- When the `selectTextCallback` asynchronous callback function that's passed as the *callback* argument of the `getSelectedDataAsync` method executes, it gets the value of the selected text when the callback returns. It gets that value from the callback's *selectedText* argument (which is of type AsyncResult) by using the value property of the returned `AsyncResult` object.

- The rest of the code in the `selectedTextCallback` function queries the XML web service for definitions.
- The remaining code in the .js file displays the list of definitions in the add-in's HTML UI.

In the add-in's web application project in Visual Studio, you can replace the contents of the **./Home.js** file with the following sample JavaScript.

JavaScript

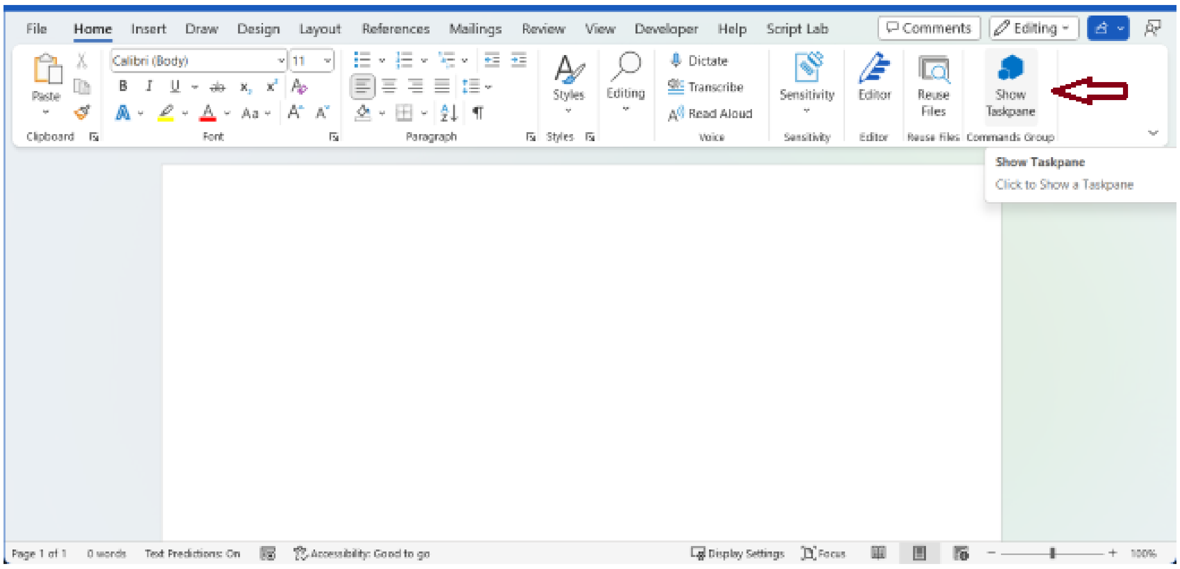
```javascript
// The document the dictionary add-in is interacting with.
let _doc;
// The last looked-up word, which is also the currently displayed word.
let lastLookup;

// The base URL for the OfficeDefinitions-conforming XML web service to
query for definitions.
const xmlServiceUrl = "DictionaryWebService.asmx/Define";

// Initialize the add-in.
// Office.initialize or Office.onReady is required for all add-ins.
Office.initialize = function (reason) {
    // Checks for the DOM to load using the jQuery ready method.
    $(document).ready(function () {
        // After the DOM is loaded, app-specific code can run.
        // Store a reference to the current document.
        _doc = Office.context.document;
        // Check whether text is already selected.
        tryUpdatingSelectedWord();
        // Add a handler to refresh when the user changes selection.
        _doc.addHandlerAsync("documentSelectionChanged",
tryUpdatingSelectedWord);
    });
}

// Executes when event is raised on the user's selection changes, and at
initialization time.
// Gets the current selection and passes that to asynchronous callback
function.
function tryUpdatingSelectedWord() {
    _doc.getSelectedDataAsync(Office.CoercionType.Text,
selectedTextCallback);
}

// Async callback that executes when the add-in gets the user's selection.
Determines whether anything should
// be done. If so, it makes requests that will be passed to various
functions.
function selectedTextCallback(selectedText) {
    selectedText = $.trim(selectedText.value);
    // Be sure user has selected text. The SelectionChanged event is raised
every time the user moves
```

```
    // the cursor, even if no selection.
    if (selectedText != "") {
        // Check whether the user selected the same word the pane is
currently displaying to
        // avoid unnecessary web calls.
        if (selectedText != lastLookup) {
            // Update the lastLookup variable.
            lastLookup = selectedText;
            // Set the "headword" span to the word you looked up.
            $("#headword").text("Selected text: " + selectedText);
            // AJAX request to get definitions for the selected word; pass
that to refreshDefinitions.
            $.ajax(xmlServiceUrl,
                {
                    data: { word: selectedText },
                    dataType: 'xml',
                    success: refreshDefinitions,
                    error: errorHandler
                });
    }
}

// This function is called when the add-in gets back the definitions target
word.
// It removes the old definitions and replaces them with the definitions for
the current word.
// It also sets the "See More" link.
function refreshDefinitions(data, textStatus, jqXHR) {
    $(".definition").remove();

    // Make a new list item for each returned definition that was returned,
set the CSS class,
    // and append it to the definitions div.
    $(data).find("Definition").each(function () {
        $(document.createElement("li"))
            .text($(this).text())
            .addClass("definition")
            .appendTo($("#definitions"));
    });

    // Change the "See More" link to direct to the correct URL.
    $("#SeeMoreLink").attr("href", $(data).find("SeeMoreURL").text());
}

// Basic error handler that writes to a div with id='message'.
function errorHandler(jqXHR, textStatus, errorThrown) {
    document.getElementById('message').innerText
      += ("textStatus:- " + textStatus
          + "\nerrorThrown:- " + errorThrown
          + "\njqXHR:- " + JSON.stringify(jqXHR));
}
```

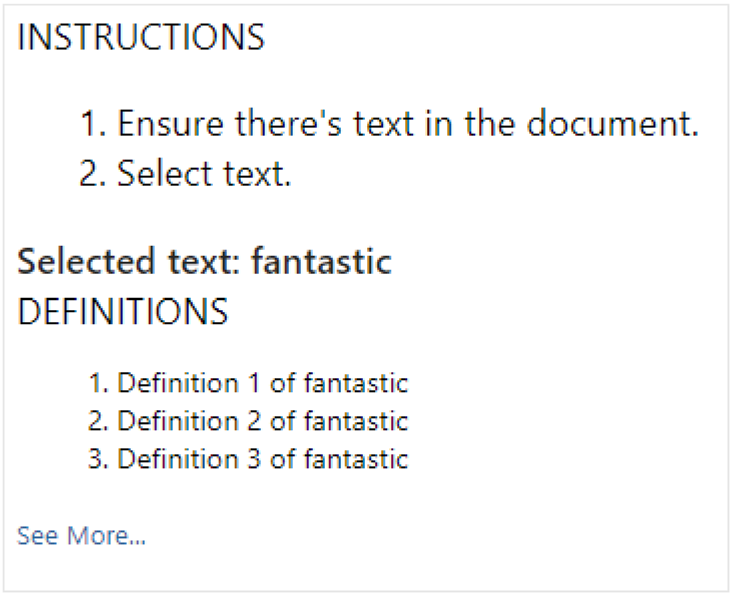
Try it out

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

2. In Word, if the add-in task pane isn't already open, choose the **Home** tab, and then choose the **Show Taskpane** button to open the add-in task pane. (If you're using the volume-licensed perpetual version of Office, instead of the Microsoft 365 version or a retail perpetual version, then custom buttons aren't supported. Instead, the task pane will open immediately.)



3. In Word, add text to the document then select any or all of that text.



INSTRUCTIONS

1. Ensure there's text in the document.
2. Select text.

Selected text: fantastic

DEFINITIONS

1. Definition 1 of fantastic
2. Definition 2 of fantastic
3. Definition 3 of fantastic

See More...

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

```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

```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

   ```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.

JavaScript

```javascript
// 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

```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

```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.");
        }
    });
}
```