Debug add-ins using developer tools in Microsoft Edge (Chromium-based)

Article • 07/14/2024

This article shows how to debug the client-side code (JavaScript or TypeScript) of your add-in when the following conditions are met.

- You can't, or don't wish to, debug using tools built into your IDE; or you are encountering a problem that only occurs when the add-in is run outside the IDE.
- Your computer is using a combination of Windows and Office versions that use the Edge (Chromium-based) webview control, WebView2.

> **Tip**
>
> For information about debugging with Edge WebView2 (Chromium-based) inside Visual Studio Code, see **Debug add-ins on Windows using Visual Studio Code and Microsoft Edge WebView2 (Chromium-based)**.

To determine which webview you're using, see Browsers and webview controls used by Office Add-ins.

> **Tip**
>
> In recent versions of Office, one way to identify the webview control that Office is using is through the **personality menu** on any add-in where it's available. (The personality menu isn't supported in Outlook.) Open the menu and select **Security Info**. In the **Security Info** dialog on Windows, the **Runtime** reports **Microsoft Edge**, **Microsoft Edge Legacy**, or **Internet Explorer**. The runtime isn't included on the dialog in older versions of Office.

Debug a task pane add-in using Microsoft Edge (Chromium-based) developer tools

> **Note**
>
> If your add-in has an **add-in command** that executes a function, the function runs in a hidden browser runtime process that the Microsoft Edge (Chromium-based)

developer tools can't be launched from, so the technique described in this article can't be used to debug code in the function.

1. Sideload and run the add-in.

   > **Note**
   >
   > To sideload an add-in in Outlook, see Sideload Outlook add-ins for testing.

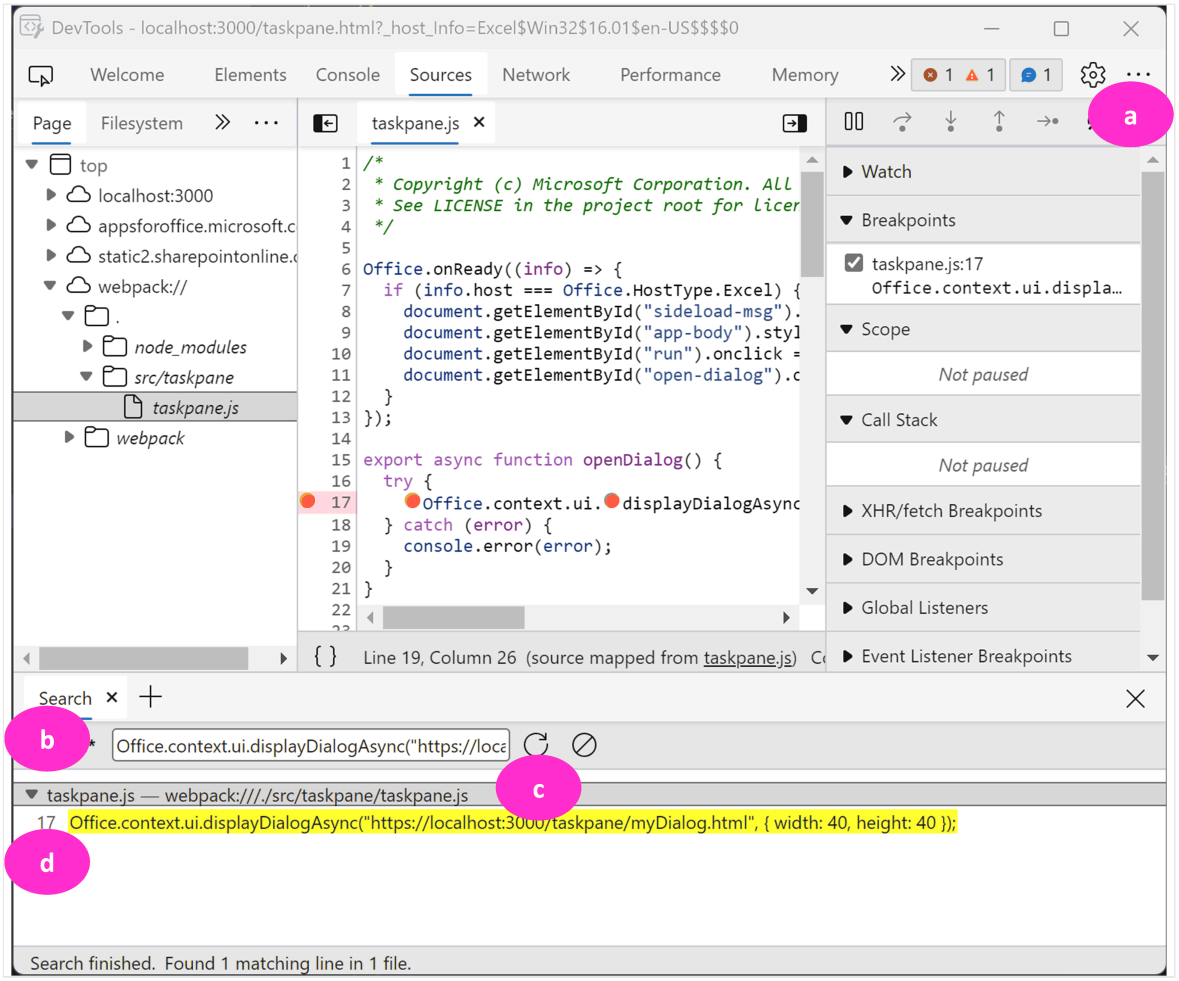
2. Run the Microsoft Edge (Chromium-based) developer tools by one of these methods:

   - Be sure the add-in's task pane has focus and press `Ctrl` + `Shift` + `I`.
   - Right-click (or select and hold) the task pane to open the context menu and select **Inspect**, or open the personality menu and select **Attach Debugger**. (The personality menu isn't supported in Outlook.)

   > **Note**
   >
   > The new Outlook on Window desktop client (preview) doesn't support the context menu or the keyboard shortcut to access the Microsoft Edge developer tools. Instead, you must run `olk.exe --devtools` from a command prompt. For more information, see the "Debug your add-in" section of Develop Outlook add-ins for the new Outlook on Windows.

3. Open the **Sources** tab.

4. Open the file that you want to debug with the following steps.
   a. On the far right of the tool's top menu bar, select the **...** button and then select **Search**.
   b. Enter a line of code from the file you want to debug in the search box. It should be something that's not likely to be in any other file.
   c. Select the refresh button.
   d. In the search results, select the line to open the code file in the pane above the search results.



5. To set a breakpoint, select the line number of the line in the code file. A red dot appears by the line in the code file. In the debugger window to the right, the breakpoint is registered in the **Breakpoints** drop down.

6. Execute functions in the add-in as needed to trigger the breakpoint.

> **Tip**
>
> For more information about using the tools, see Microsoft Edge Developer Tools overview.

Debug a dialog in an add-in

If your add-in uses the Office Dialog API, the dialog runs in a separate process from the task pane (if any) and the tool must be started from that separate process. Follow these steps.

1. Run the add-in.

2. Open the dialog and be sure it has focus.

3. Open the Microsoft Edge (Chromium-based) developer tools by one of these methods:

    - Press `Ctrl` + `Shift` + `I` or `F12`.
    - Right-click (or select and hold) the dialog to open the context menu and select **Inspect**.

4. Use the tool the same as you would for code in a task pane. See Debug a task pane add-in using Microsoft Edge (Chromium-based) developer tools earlier in this article.

Debug Office Add-ins on Windows using Visual Studio Code and Microsoft Edge WebView2 (Chromium-based)

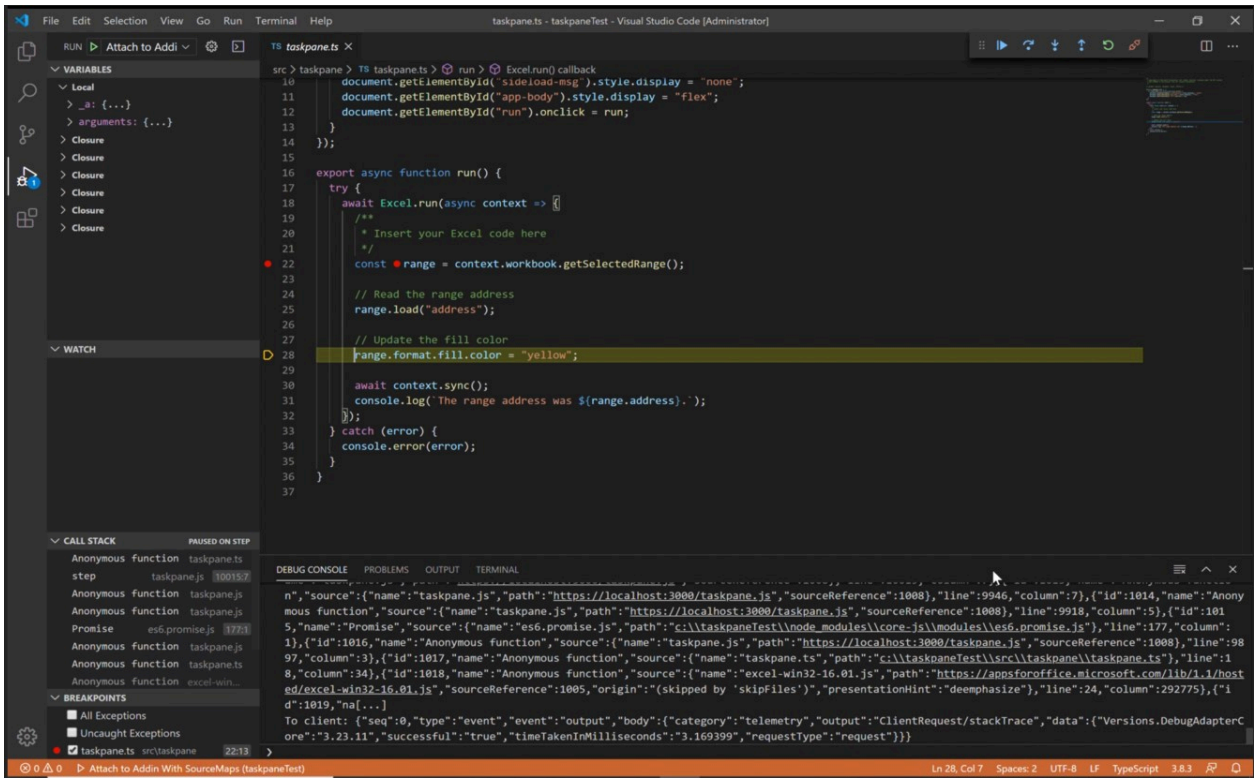
Article • 04/15/2024

Office Add-ins running on Windows can debug against the Edge Chromium WebView2 runtime directly in Visual Studio Code.

> **Important**
>
> This article only applies when Office runs add-ins in the Microsoft Edge Chromium WebView2 runtime, as explained in Browsers and webview controls used by Office Add-ins. For instructions about debugging in Visual Studio Code against Microsoft Edge Legacy with the original WebView (EdgeHTML) runtime, see Debug add-ins using developer tools in Microsoft Edge Legacy.

> **Tip**
>
> If you can't, or don't wish to, debug using tools built into Visual Studio Code; or you're encountering a problem that only occurs when the add-in is run outside Visual Studio Code, you can debug Edge Chromium WebView2 runtime by using the Edge (Chromium-based) developer tools as described in Debug add-ins using developer tools for Microsoft Edge WebView2.

This debugging mode is dynamic, allowing you to set breakpoints while code is running. See changes in your code immediately while the debugger is attached, all without losing your debugging session. Your code changes also persist, so you see the results of multiple changes to your code. The following image shows this extension in action.



Prerequisites

- Visual Studio Code
- Node.js (version 10+)
- Windows 10, 11
- A combination of platform and Office application that supports Microsoft Edge with WebView2 (Chromium-based) as explained in Browsers and webview controls used by Office Add-ins. If your version of Office from a Microsoft 365 subscription is earlier than Version 2101, you'll need to install WebView2. For instructions to install WebView2, see Microsoft Edge WebView2 / Embed web content ... with Microsoft Edge WebView2.

Debug a project created with Yo Office

These instructions assume you have experience using the command line, understand basic JavaScript, and have created an Office Add-in project before using the Yeoman generator for Office Add-ins. If you haven't done this before, consider visiting one of our tutorials, such as the Excel Office Add-in tutorial.

1. The first step depends on the project and how it was created.

    - If you want to create a project to experiment with debugging in Visual Studio Code, use the Yeoman generator for Office Add-ins. Follow any of the Yo Office quick start guides, such as the Outlook add-in quick start.

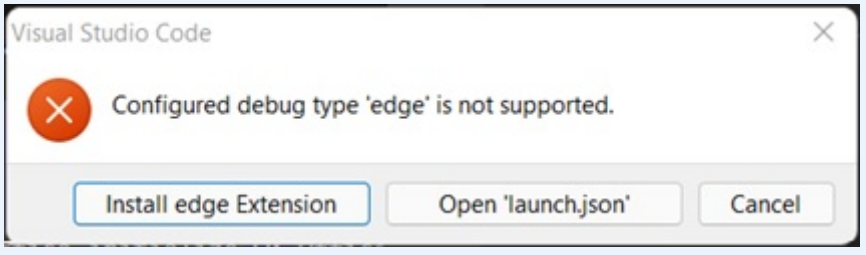
- If you want to debug an existing project that was created with Yo Office, skip to the next step.

2. Open VS Code and open your project in it.

3. Choose **View** > **Run** or enter `Ctrl` + `Shift` + `D` to switch to debug view.

4. From the **RUN AND DEBUG** options, choose the Edge Chromium option for your host application, such as **Outlook Desktop (Edge Chromium)**. Select `F5` or choose **Run** > **Start Debugging** from the menu to begin debugging. This action automatically launches a local server in a Node window to host your add-in and then automatically opens the host application, such as Excel or Word. This may take several seconds.

   > **Tip**
   >
   > If you aren't using a project created with Yo Office, you may be prompted to adjust a registry key. While in the root folder of your project, run the following in the command line.
   >
   > command line
   >
   > ```
   > npx office-addin-debugging start <your manifest path>
   > ```

   > **Important**
   >
   > If your project was created with older versions of Yo Office, you may see the following error dialog box about 10 - 30 seconds after you start debugging (at which point you may have already gone on to another step in this procedure) and it may be hidden behind the dialog box described in the next step.
   >
   > 
   >
   >
   > Complete the tasks in the Appendix and then restart this procedure.

5. In the host application, your add-in is now ready to use. Select **Show Taskpane** or run any other add-in command. A dialog box will appear with text similar to the

following:

> WebView Stop On Load. To debug the webview, attach VS Code to the webview instance using the Microsoft Debugger for Edge extension, and click OK to continue. To prevent this dialog from appearing in the future, click Cancel.

Select **OK**.

> ⓘ Note
>
> If you select **Cancel**, the dialog won't be shown again while this instance of the add-in is running. However, if you restart your add-in, you'll see the dialog again.

6. You're now able to set breakpoints in your project's code and debug. To set breakpoints in Visual Studio Code, hover next to a line of code and select the red circle that appears.

```
16  export async function run() {
17    try {
18      await Excel.run(async context => {
19        /**
20         * Insert your Excel code here
21         */
22        const range = context.workbook.getSelectedRange();
23
24        // Read the range address
25        range.load("address");
26
27        // Update the fill color
28        range.format.fill.color = "yellow";
29
30        await context.sync();
31        console.log(`The range address was ${range.address}.`);
32      });
33    } catch (error) {
34      console.error(error);
35    }
36  }
37
```

7. Run functionality in your add-in that calls the lines with breakpoints. You'll see that breakpoints have been hit and you can inspect local variables.

> **ⓘ Note**
>
> Breakpoints in calls of `Office.initialize` or `Office.onReady` are ignored. For details about these functions, see Initialize your Office Add-in.

> **ⓘ Important**
>
> The best way to stop a debugging session is to select `Shift` + `F5` or choose **Run** > **Stop Debugging** from the menu. This action should close the Node server window and attempt to close the host application, but there'll be a prompt on the host application asking you whether to save the document or not. Make an appropriate choice and let the host application close. Avoid manually closing the Node window or host application. Doing so can cause bugs especially when you are stopping and starting debugging sessions repeatedly.
>
> If debugging stops working---for example, if breakpoints are being ignored---stop debugging. Then, if necessary, close all host application windows and the Node window. Finally, close Visual Studio Code and reopen it.

Debug a project not created with Yo Office

If your project wasn't created with Yo Office, you need to create a debug configuration for Visual Studio Code.

Configure package.json file

1. Ensure you have a `package.json` file. If you don't already have a package.json file, run `npm init` in the root folder of your project and answer the prompts.
2. Run `npm install office-addin-debugging`. This package sideloads your add-in for debugging.
3. Open the `package.json` file. In the `scripts` section, add the following script.

```JSON
"start:desktop": "office-addin-debugging start $MANIFEST_FILE$
desktop",
"dev-server": "$SERVER_START$"
```

4. Replace `$MANIFEST_FILE$` with the correct file name and folder location of your manifest.

5. Replace `$SERVER_START$` with the command to start your web server. Later in these steps, the `office-addin-debugging` package will specifically look for the `dev-server` script to launch your web server.

6. Save and close the `package.json` file.

Configure launch.json file

1. Create a file named `launch.json` in the `\.vscode` folder of the project if there isn't one there already.

2. Copy the following JSON into the file.

```JSON
{
  // Other properties may be here.
  "configurations": [
    {
      "name": "$HOST$ Desktop (Edge Chromium)",
      "type": "msedge",
      "request": "attach",
      "useWebView": true,
      "port": 9229,
      "timeout": 600000,
      "webRoot": "${workspaceRoot}",
      "preLaunchTask": "Debug: Excel Desktop"
    }
  ]
  // Other properties may be here.
}
```

> **Note**
>
> If you already have a `launch.json` file, just add the single configuration to the `configurations` section.

3. Replace the placeholder `$HOST$` with the name of the Office application that the add-in runs in. For example, `Outlook` or `Word`.

4. Save and close the file.

Configure tasks.json

1. Create a file named `tasks.json` in the `\.vscode` folder of the project.
2. Copy the following JSON into the file. It contains a task that starts debugging for your add-in.

   JSON

   ```
   {
     "version": "2.0.0",
     "tasks": [
       {
         "label": "Debug: $HOST$ Desktop",
         "type": "shell",
         "command": "npm",
         "args": ["run", "start:desktop", "--", "--app", "$HOST$"],
         "presentation": {
           "clear": true,
           "panel": "dedicated"
         },
         "problemMatcher": []
       }
     ]
   }
   ```

   > **Note**
   >
   > If you already have a `tasks.json` file, just add the single task to the `tasks` section.

3. Replace both instances of the placeholder `$HOST$` with the name of the Office application that the add-in runs in. For example, `Outlook` or `Word`.

You can now debug your project using the VS Code debugger (F5).

Appendix

1. In the error dialog box, select the **Cancel** button.
2. If debugging doesn't stop automatically, select `Shift` + `F5` or choose **Run** > **Stop Debugging** from the menu.
3. Close the Node window where the local server is running, if it doesn't close automatically.
4. Close the Office application if it doesn't close automatically.
5. Open the `\.vscode\launch.json` file in the project.

6. In the `configurations` array, there are several configuration objects. Find the one whose name has the pattern `$HOST$ Desktop (Edge Chromium)`, where $HOST$ is an Office application that your add-in runs in; for example, `Outlook Desktop (Edge Chromium)` or `Word Desktop (Edge Chromium)`.
7. Change the value of the `"type"` property from `"edge"` to `"pwa-msedge"`.
8. Change the value of the `"useWebView"` property from the string `"advanced"` to the boolean `true` (note there are no quotation marks around the `true`).
9. Save the file.
10. Close VS Code.

See also

- Test and debug Office Add-ins
- Debug add-ins using developer tools for Internet Explorer
- Debug add-ins using developer tools for Edge Legacy
- Debug add-ins using developer tools in Microsoft Edge (Chromium-based)
- Attach a debugger from the task pane
- Runtimes in Office Add-ins