

Test Office Add-ins

Article • 07/12/2024

This article contains guidance about testing, debugging, and troubleshooting issues with Office Add-ins.

Test cross-platform and for multiple versions of Office

Office Add-ins run across major platforms, so you need to test an add-in in all the platforms where your users might be running Office. This usually includes Office on the web, Office on Windows (both perpetual and Microsoft 365 subscription), Office on Mac, Office on iOS, and (for Outlook add-ins) Office on Android. However, there may be some situations in which you can be sure that none of your users will be working on some platforms. For example, if you're making an add-in for a company that requires its users to work with Windows computers and subscription Office, then you don't need to test for Office on Mac or perpetual Office on Windows.

ⓘ Note

On Windows computers, the version of Windows and Office will determine which browser or webview control is used by add-ins. For more information, see [Browsers and webview controls used by Office Add-ins](#). For brevity hereafter, this article uses "browser control" to mean "browser or webview control".

Add-ins tested for Office on the web

Add-ins are tested for Office on the web with all major modern browsers, including Microsoft Edge (Chromium-based WebView2), Chrome, and Safari. Accordingly, you should test on these platforms and browsers before you submit to [AppSource](#). For more information about validation, see [Commercial marketplace certification policies](#), especially [section 1120.3](#), and the [Office Add-in application and availability page](#).

Office on the web no longer opens in Internet Explorer or Microsoft Edge Legacy (EdgeHTML). Consequently, AppSource doesn't test Office on the web on these browsers. Office still supports these browsers for add-in runtimes, so if you think you've encountered a bug in how add-ins run in them, please create an issue in the [office-js](#) ↗

repository. For more information, see [Support older Microsoft webviews and Office versions](#) and [Troubleshoot EdgeHTML and WebView2 \(Microsoft Edge\) issues](#).

Add-ins tested for Office on Windows

Some Office versions on Windows still use the webview controls that come with Internet Explorer and Microsoft Edge Legacy. AppSource tests whether your add-in supports these browser controls. If your add-in doesn't support these browser controls, AppSource only issues a warning and doesn't reject your add-in. In this instance, we recommend configuring a graceful failure message on your add-in for a smoother user experience. For further guidance, see [Support older Microsoft webviews and Office versions](#).

Sideload an Office Add-in for testing

You can use sideloading to install an Office Add-in for testing without having to first put it in an add-in catalog. The procedure for sideloading an add-in varies by platform, and in some cases, by product as well. The following articles each describe how to sideload Office Add-ins on a specific platform or within a specific product.

ⓘ Note

Office Add-ins that use the unified manifest for Microsoft 365 are *directly* supported in Office on the web, in [new Outlook on Windows](#) [↗], and in Office on Windows connected to a Microsoft 365 subscription, Version 2304 (Build 16320.00000) or later. When the app package that contains the unified manifest is sideloaded to a platform that doesn't directly support that type of manifest then an add-in only manifest is generated from the unified manifest and this manifest is the one that's sideloaded.

- [Sideload Office Add-ins in Office on the web](#)
- [Sideload Office Add-ins on Windows](#)
- [Sideload Office Add-ins on Mac](#)
- [Sideload Office Add-ins on iPad](#)
- [Sideload Outlook add-ins for testing](#)

Unit testing

For information about how to add unit tests to your add-in project, see [Unit testing in Office Add-ins](#).

Debug an Office Add-in

The procedure for debugging an Office Add-in varies based on your platform and environment. For more information, see [Debug Office Add-ins](#).

Validate an Office Add-in manifest

For information about how to validate the manifest file that describes your Office Add-in and troubleshoot issues with the manifest file, see [Validate and troubleshoot issues with your manifest](#).

Troubleshoot user errors

For information about how to resolve common issues that users may encounter with your Office Add-in, see [Troubleshoot user errors with Office Add-ins](#).

Overview of debugging Office Add-ins

Article • 04/02/2025

Debugging Office Add-ins is essentially the same as debugging any web application. However, a single set of tools won't work for all add-in developers. This is because add-ins can be developed on different operating systems and run cross-platform. This article helps you find the detailed debugging guidance for your development environment.

Tip





This article is concerned with debugging in the narrow sense of setting breakpoints and stepping through code. For guidance on testing and troubleshooting, start with [Test Office Add-ins](#) and [Troubleshoot development errors with Office Add-ins](#).

Note

Although you should *test* your add-in on all the platforms that you want to support, you'll only very rarely need to *debug* on an environment different from your development computer. For this reason, this article uses "your development computer" and "your development environment" to refer to the environment on which you're debugging. If a problem in the code occurs only on a platform other than the one on your development computer, and you need to set breakpoints or step through code to solve it, then the environment on which you're debugging isn't literally your development environment.

Server-side or client-side?

Debugging the server-side code of an Office Add-in is the same as debugging the server-side of any web application. See the debugging instructions for your IDE or other tools. The following are examples for some of the most popular tools.

- [Debug ASP.NET or ASP.NET Core apps in Visual Studio](#)
- [Debugging Express](#) 
- [Node.js Debugging Guide](#) 
- [Node.js debugging in VS Code](#) 
- [Webpack Debugging](#) 

The rest of this article is concerned only with debugging client-side JavaScript (which may be transpiled from TypeScript).

Special cases

There are some special cases in which the debugging process differs from normal for a given combination of platform, Office application, and development environment. If you're debugging any of these special cases, use the links in this section to find the proper guidance. Otherwise, continue to [General guidance](#).

- Debugging the `Office.initialize` or `Office.onReady` function: [Debug the initialize and onReady functions](#).
- Debugging an Excel custom function in a *non-shared* runtime: [Custom functions debugging in a non-shared runtime](#).
- Debugging a **function command** in a *non-shared* runtime:
 - Outlook add-ins on a Windows development computer: [Debug function commands in Outlook add-ins](#)
 - Other Office application add-ins or Outlook on a Mac development computer: [Debug a function command with a non-shared runtime](#).
- Debugging an event-based or spam-reporting Outlook add-in: [Debug event-based and spam-reporting add-ins](#).
- Debugging an add-in in the new Outlook on Windows desktop client (preview): See the "Debug your add-in" section of [Develop Outlook add-ins for the new Outlook on Windows](#).
- Debugging a Blazor-based add-in: Debug the add-in the same way you would debug a Blazor web application. See [Debug ASP.NET Core Blazor WebAssembly](#).

General guidance

To find guidance for debugging client-side code, the first variable is the operating system of your development computer.

- [Windows](#)
- [Mac](#)
- [Linux or other Unix variant](#)

Debug on Windows

The following provides general guidance to debugging on Windows. Debugging on Windows depends on your IDE.

- **Visual Studio:** Debug using the browser's F12 tools. See [Debug Office Add-ins in Visual Studio](#).
- **Any other IDE** (or you don't want to debug inside your IDE): Use the developer tools that are associated with the webview control that add-ins use on your development computer. See one of the following:
 - For the Trident webview: [Debug add-ins using developer tools for Internet Explorer](#)
 - For the EdgeHTML webview: [Debug add-ins using developer tools for Edge Legacy](#)
 - For the WebView2 webview: [Debug add-ins using developer tools in Microsoft Edge \(Chromium-based\)](#)

For information about which runtime is being used, see [Browsers and webview controls used by Office Add-ins](#) and [Runtimes in 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 on Mac

Use the Safari Web Inspector. Instructions are in [Debug Office Add-ins on a Mac](#).

Debug on Linux

There is no desktop version of Office for Linux, so you'll need to [sideload the add-in to Office on the web](#) to test and debug it. Debugging guidance is in [Debug add-ins in Office on the web](#).

Note

We don't recommend that you develop Office Add-ins on a Linux computer except in the unusual case where you can be sure that all the add-in's users will be accessing the add-in through Office on the web from a Linux computer.

Debug add-ins in staging or production

To debug an add-in that is already in staging or production, attach a debugger from the UI of the add-in. For instructions, see [Attach a debugger from the task pane](#).

Versions of office.js for debugging

There are debug versions of the Office JavaScript libraries. These versions are more human readable and easier to step through with a debugger. Use them when the Office JavaScript APIs aren't working as expected. Avoid using them when you publish and deploy your add-in.

The debug versions are found at the following CDN locations.

- Office JavaScript API library:

```
https://appsforoffice.microsoft.com/lib/1/hosted/office.debug.js
```

- Office JavaScript API (preview) library:

```
https://appsforoffice.microsoft.com/lib/beta/hosted/office.debug.js
```

See also

- [Runtimes in Office Add-ins](#)

Sideload Office Add-ins that use the unified manifest for Microsoft 365

08/13/2025

The process of sideloading an add-in that uses the [Unified manifest for Microsoft 365](#) varies depending on the tool you want to use and on how the add-in project was created.

ⓘ Note

An add-in that uses the unified manifest can be sideloaded on Office on Windows, Version 2304 (Build 16320.20000) or later. Sideloaded on Windows has the effect of sideloading to Office on the web too. Currently, it can't be sideloaded on Mac or iPad.

Sideload add-ins created with the Yeoman generator for Office Add-ins (Yo Office)

Use the process described in [Sideload with a system prompt, bash shell, or terminal](#).

Sideload with Microsoft 365 Agents Toolkit

1. First, *make sure Office desktop application that you want to sideload into is closed*.
2. In Visual Studio Code, open Agents Toolkit.
3. Required for Outlook only: in the **ACCOUNTS** section, verify that you're signed into Microsoft 365.
4. Select **View | Run** in Visual Studio Code. In the **RUN AND DEBUG** dropdown menu, select one of these options as appropriate for your add-in.
 - Excel Desktop (Edge Chromium)
 - Outlook Desktop (Edge Chromium)
 - PowerPoint Desktop (Edge Chromium)
 - Word Desktop (Edge Chromium)
5. Press **F5**. The project builds and a Node dev-server window opens. This process may take a couple of minutes and then the desktop version of the Office application that you selected opens. You can now work with your add-in. For an Outlook add-in, be sure you're working in the **Inbox** of *your Microsoft 365 account identity*.

6. To stop debugging and uninstall the add-in, select **Run | Stop Debugging** in Visual Studio Code. Closing the server window doesn't reliably stop the server and closing the Office application doesn't reliably cause Office to unacquire the add-in.

⚠ Note

If the preceding step seems to have no effect, uninstall the add-in by opening a **TERMINAL** in Visual Studio Code, and then complete the uninstall step — the *last* step — of the section [Sideload with a system prompt, bash shell, or terminal](#).

Sideload with a system prompt, bash shell, or terminal

1. First, *make sure the Office desktop application that you want to sideload into is closed*.
2. Open a system prompt, bash shell, or the Visual Studio Code **TERMINAL**, and navigate to the root of the project.
3. The command to sideload the add-in depends on when the project was created. If the `"scripts"` section of the project's package.json file has a `"start:desktop"` script, then run `npm run start:desktop`; otherwise, run `npm run start`. The project builds and a Node dev-server window opens. This process may take a couple of minutes then the Office host application (Excel, Outlook, PowerPoint, or Word) desktop opens.
4. On some versions of Office, the add-in may not fully activate. For example, the add-in's buttons may not appear on the ribbon. If this happens, select the **Add-ins** button on the **Home** ribbon. On the flyout that opens, select the add-in. This completes the installation.
5. You can now work with your add-in.
6. When you're done working with your add-in, make sure to run the command `npm run stop`. Closing the server window doesn't reliably stop the server and closing the Office application doesn't reliably cause Office to unacquire the add-in.

Sideload other NodeJS and npm projects

There are two tools you can use to sideload.

Sideload with the Office-Addin-Debugging tool

1. To sideload the add-in, run the following command. This command puts the unified manifest and the two icon image files that are referenced in the manifest's `"icons"` property into a zip file and sideloads it to the Office application. It also starts a server in a

separate NodeJS window to host the add-in files on localhost. For more details about this command, see [Office-Addin-Debugging](#).

command line

```
npx office-addin-debugging start <relative-path-to-unified-manifest> desktop
```

2. When you use office-addin-debugging to start an add-in, *always stop the session with the following command*. Closing the server window doesn't reliably stop the server and closing the Office application doesn't reliably cause Office to unacquire the add-in.

command line

```
npx office-addin-debugging stop <relative-path-to-unified-manifest>
```

Sideload with Microsoft 365 Agents Toolkit CLI (command-line interface)

1. Create a zip package. See [Manually create the add-in package file](#).
2. In the root of the project, open a command prompt or bash shell and run the following command to install the Agents Toolkit CLI.

command line

```
npm install -g @microsoft/m365agentstoolkit-cli
```

3. Run the following command to sideload the add-in.

command line

```
atk install --file-path <relative-path-to-zip-file>
```

Important

This command returns some information about the add-in including an autogenerated title ID as shown in the following example.

```
>atk install --file-path manifests/contoso/contoso.zip
```

```
Using account admin@██████████.com  
TitleId: U_5372174d-00e6-53bf-40e3-9099abd1f680  
AppId: 805eef62-61ce-43f9-be32-1ad5f954093e
```

You'll need this title ID to end the sideloading and debugging session. It is recorded on Windows computers in the following Registry key:

**HKEY_CURRENT_USER\SOFTWARE\Microsoft\Office\16.0\Wef\Developer\Outlook
SideloadManifestPath\TitleId**

The string "Outlook" is in the key name for historical reasons, but it applies to any add-in installed with the Agents Toolkit CLI.

Only the most recent add-in installed with the CLI is recorded. If you sideload an add-in with the CLI before you have uninstalled an earlier add-in you installed with the CLI, then there is no record of the earlier add-in's title ID in the Registry. So, we recommend that you also save it in a text file in the root of the project and name the file **TitleID.txt** on both Mac and Windows computers.

4. When you use the Agents Toolkit CLI to start an add-in, *always stop the session with the following command*. Closing the server window doesn't reliably stop the server and closing the Office application doesn't reliably cause Office to unacquire the add-in. Replace "{title ID}" with the title ID of the add-in including the "U_" prefix; for example, `U_90d141c6-cf4f-40ee-b714-9df9ea593f39`.

command line

```
atk uninstall --mode title-id --title-id {title ID} --interactive false
```

Important

The [documentation for the uninstall command](#) describes a way to use the add-in's manifest ID instead of the title ID. Due to a bug in an API that the CLI calls, this option doesn't currently work. You must use the `uninstall` command given above and you must include the `--interactive false` option.

Sideload through the Teams app store

Add-ins that use the unified manifest can be manually sideloaded through the Teams app store, even if they have no Teams-related functionality. The steps are as follows.

1. Create an app package manually if it hasn't already been created by a tool. See [Manually create the add-in package file](#).
 2. Close all Office applications, and then clear the Office cache following the instructions at [Manually clear the cache](#).
 3. Open Teams and select **Apps** from the app bar, then select **Manage your apps** at the bottom of the **Apps** pane.
 4. Select **Upload an app** in the **Apps** dialog, and then in the dialog that opens, select **Upload a custom app**.
 5. In the **Open** dialog, navigate to, and select, the app package.
 6. Select **Add** in the dialog that opens.
 7. When you're prompted that the app was added, *don't* open it in Teams. Instead, close Teams.
 8. The next task is to start a local web server that hosts your project's HTML and JavaScript files. How you do this depends on several factors including the folder structure of your project, the tools you use, such as a bundler, task manager, server application, and how you have configured those tools. The following instruction applies only to projects that meet the following conditions.
 - There's a **webpack.config.js** file in the root of the project that is similar to the ones in add-in projects that are created with the [Yeoman Generator for Office Add-ins](#) or [Microsoft 365 Agent Toolkit](#).
 - There's a **package.json** file in the root of the project similar to the ones created by the same two tools and the file has a "scripts" section with the following script in it.
- JSON

```
"dev-server": "webpack serve --mode development"
```
9. In a command prompt or Visual Studio Code **TERMINAL** in the root of the project, run `npm run dev-server` to start the server on localhost.
10. Open the Office application that the add-in targets. Wait until the add-in has loaded. This may take as much as two minutes. Depending on your version of Office, ribbon buttons and other artifacts may appear automatically. In some versions, you need to manually

activate the add-in: Select the **Add-ins** button on the **Home** ribbon, and then in the flyout that opens, select your add-in. It will have the name specified in the `"name.short"` property of the manifest.

Important

When you want to end a testing session and make changes to the add-in that you sideloaded through the Teams app store, be sure to remove the add-in completely with the following steps.

1. Close the Office application.
2. Shut down the server. See the documentation for your server application for how to do this. For the webpack dev-server application, shutting it down depends on whether the server is running in the same window in which you ran `npm run dev-server` or a different window. If it's the same window, give the terminal focus and press `Ctrl + C`. Choose "Y" in response to the prompt to end the process. If it's in a different window, then in the window where you ran `npm run dev-server`, run `npm run stop`.
3. Clear the Office cache following the instructions at [Manually clear the cache](#).
4. Open Teams and select **Apps** from the app bar, then select **Manage your apps** at the bottom of the **Apps** pane.
5. Find your add-in in the list of apps. It will have the name specified in the `"name.short"` property of the manifest.
6. Select the add-in from the list of apps to expand its row.
7. Select the trash can icon and then select **Remove** in the prompt.

Make your changes and then sideload the add-in again.

Manually create the add-in package file

When the unified manifest is used, the unit of installation and sideloading is a zip-formatted package file. This file is usually created for you by the tools you use to create and test your add-in, but there are scenarios in which you create it manually. To do so, use any zip utility to create a zip file that contains the following files.

- The unified manifest, which goes in the root of the zip file.
- The two image files referenced in the `"icons"` property of the manifest.
- Any localization files that are referenced in the `"localizationInfo"` property of the manifest.

- Any declarative agent files that are referenced in the `"copilotAgents"` property.
- Any second-level supplementary files. For example, declarative agent configuration files sometimes reference second-level supplementary files, such as plugin configuration files. These should be included too.

Important

All of these files must have the same relative path in the zip file as specified in the manifest.

For example, if the path of the two image files is **assets/icon-64.png** and **assets/icon-128.png**, then you must include an **assets** folder with the two files in the zip package. Second-level files, such as plugin configuration files for declarative agents, must have the same relative path in the zip file as they do in the first-level file that references them. For example, if the relative path of a declarative agent file specified in the manifest is **agents/myAgent.json**, then you must include an **agents** folder in the zip package and put the **myAgent.json** file in it. If the declarative agent file, in turn, gives the relative path of **plugins/myPlugin.json** for a plugin configuration file, then you must include a **plugins** subfolder under the **agents** folder and put the **myPlugin.json** file in it.

To maximize compatibility with Microsoft 365 development tools, we recommend that you keep the files that will be included in the package in a folder called **appPackage** in the root of your project, and that you put the package file in a subfolder named **build** in the **appPackage** folder.

The following are examples of the recommended structure. The structure inside the **\build\appPackage.zip** file must mirror the structure of the **appPackage** folder, except for the **build** folder itself.

Console

```
\appPackage
  \assets
    color.png
    outline.png
  \build
    appPackage.zip
  manifest.json
```

Console

```
\appPackage
  \agents
    myAgent.json
  \plugins
    myPlugin.json
```

```
\assets
  color.png
  outline.png
\build
  appPackage.zip
\languages
  fr-FR.json
  es-MX.json
manifest.json
```

Sideload Office Add-ins for testing from a network share

Article • 05/21/2025

You can test an Office Add-in in an Office client that's on Windows by publishing the manifest to a network file share (instructions follow). This deployment option is intended to be used when you've completed development and testing on a localhost and want to test the add-in from a non-local server or cloud account.

📘 Important

Deployment by network share isn't supported for production add-ins. This method has the following limitations.

- The add-in can only be installed on Windows computers.
- Add-ins that use the [unified manifest for Microsoft 365](#) aren't supported when published to a network share.
- If a new version of an add-in changes the ribbon, such as by adding a custom tab or custom button to it, each user will have to reinstall the add-in.

⚠️ Note

If your add-in project was created with a sufficiently recent version of the [Yeoman generator for Office Add-ins](#), the add-in will automatically sideload in the Office desktop client when you run `npm start`.

This article applies only to testing Word, Excel, PowerPoint, and Project add-ins and only on Windows. If you want to test on another platform or want to test an Outlook add-in, see one of the following topics to sideload your add-in.

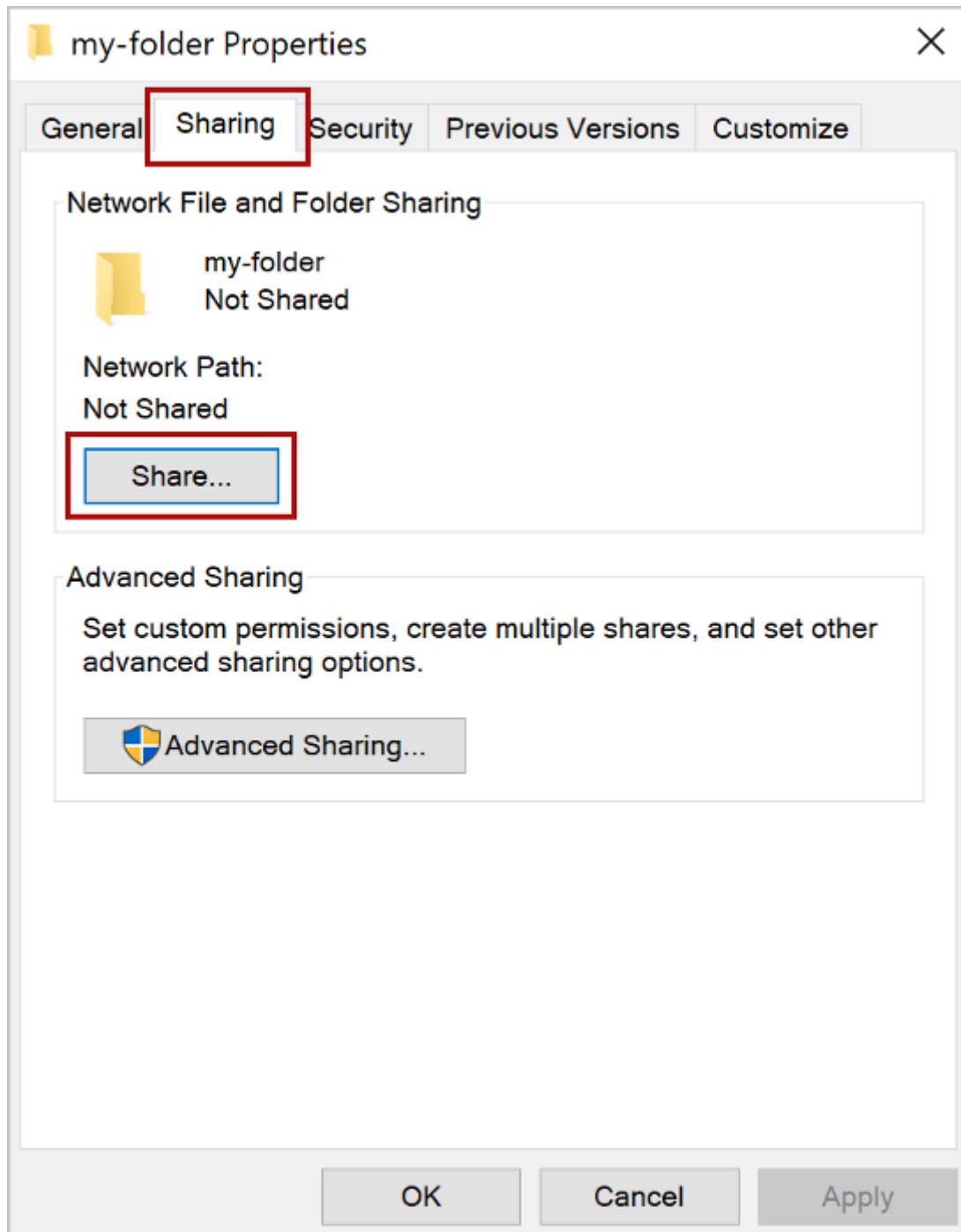
- [Sideload Office Add-ins in Office on the web for testing](#)
- [Sideload Office Add-ins on Mac for testing](#)
- [Sideload Office Add-ins on iPad for testing](#)
- [Sideload Outlook add-ins for testing](#)

The following video walks you through the process of sideloading your add-in in Office on the web or desktop using a shared folder catalog.

<https://www.youtube-nocookie.com/embed/XXsAw2UUiQo> ↗

Share a folder

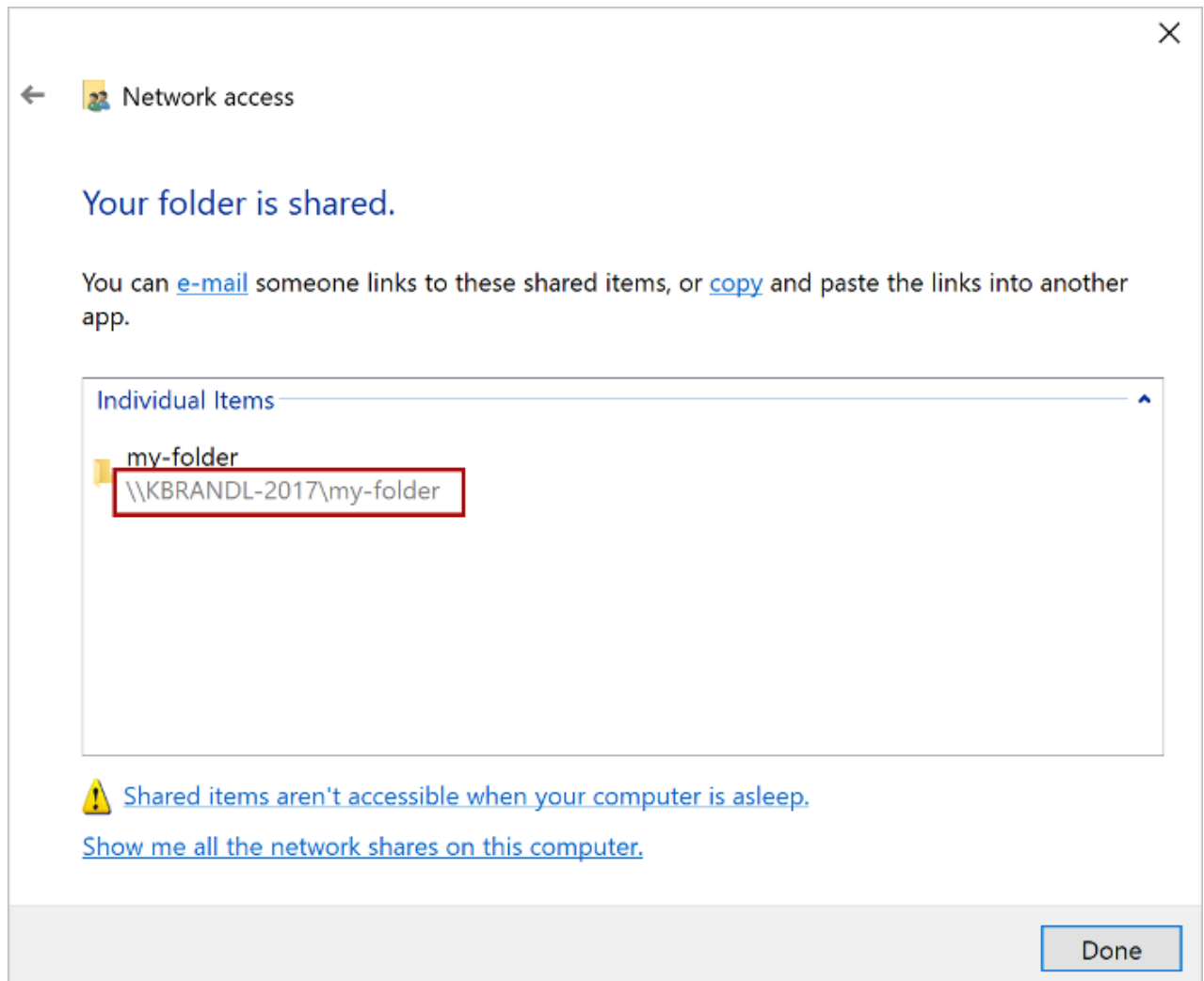
1. In File Explorer on the Windows computer where you want to host your add-in, go to the parent folder, or drive letter, of the folder you want to use as your shared folder catalog.
2. Open the context menu for the folder you want to use as your shared folder catalog (for example, right-click the folder) and choose **Properties**.
3. Within the **Properties** dialog window, open the **Sharing** tab and then choose the **Share** button.



4. Within the **Network access** dialog window, add yourself and any other users and/or groups with whom you want to share your add-in. You'll need at least **Read/Write**

permission to the folder. After you've finished choosing people to share with, choose the **Share** button.

5. When you see the **Your folder is shared** confirmation, make note of the full network path that's displayed immediately following the folder name. (You'll need to enter this value as the **Catalog Url** when you [specify the shared folder as a trusted catalog](#), as described in the next section of this article.) Choose the **Done** button to close the **Network access** dialog window.



6. Choose the **Close** button to close the **Properties** dialog window.

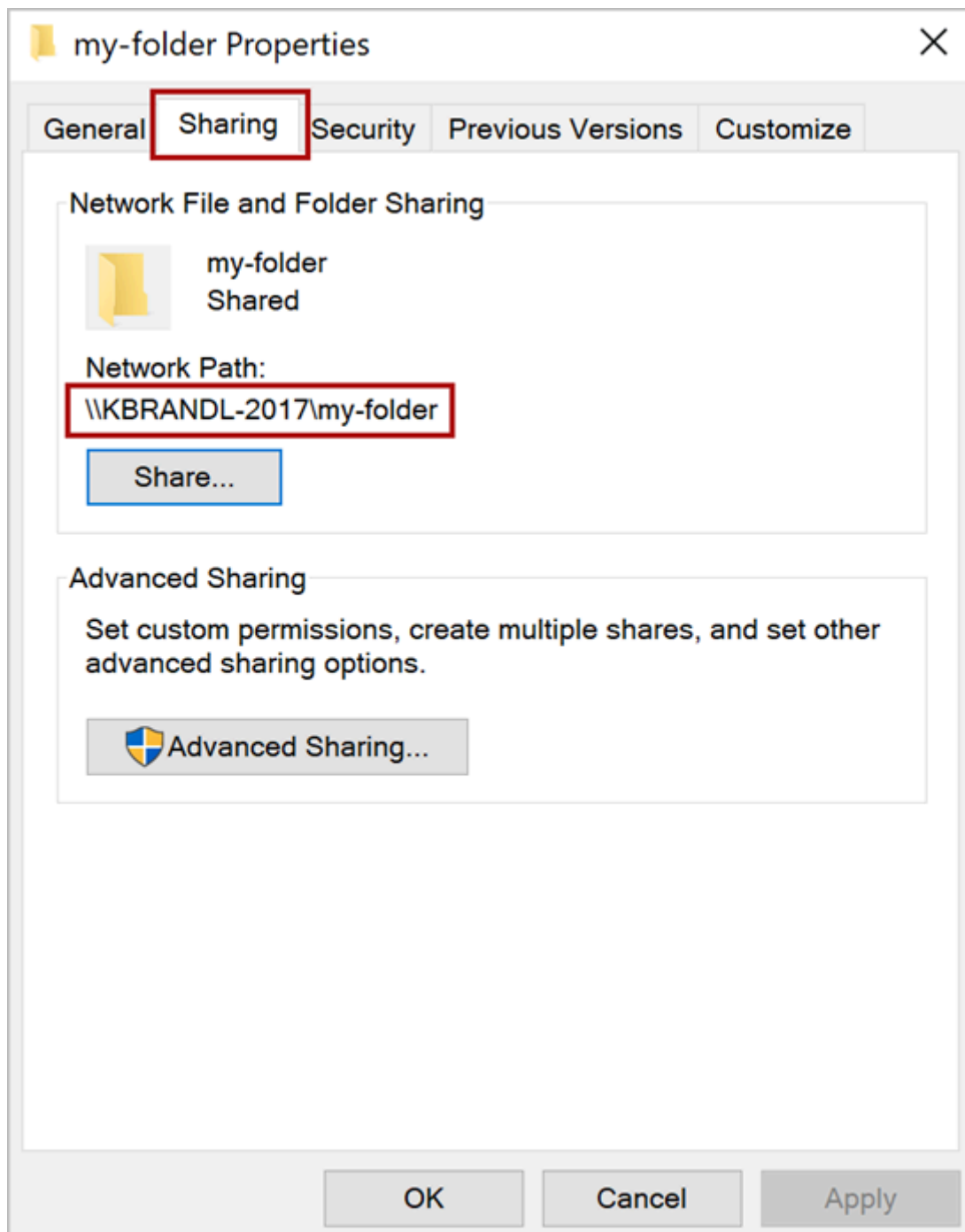
Specify the shared folder as a trusted catalog

There are two options for how you specify this trust. Follow the instructions for the option that works better for your setup.

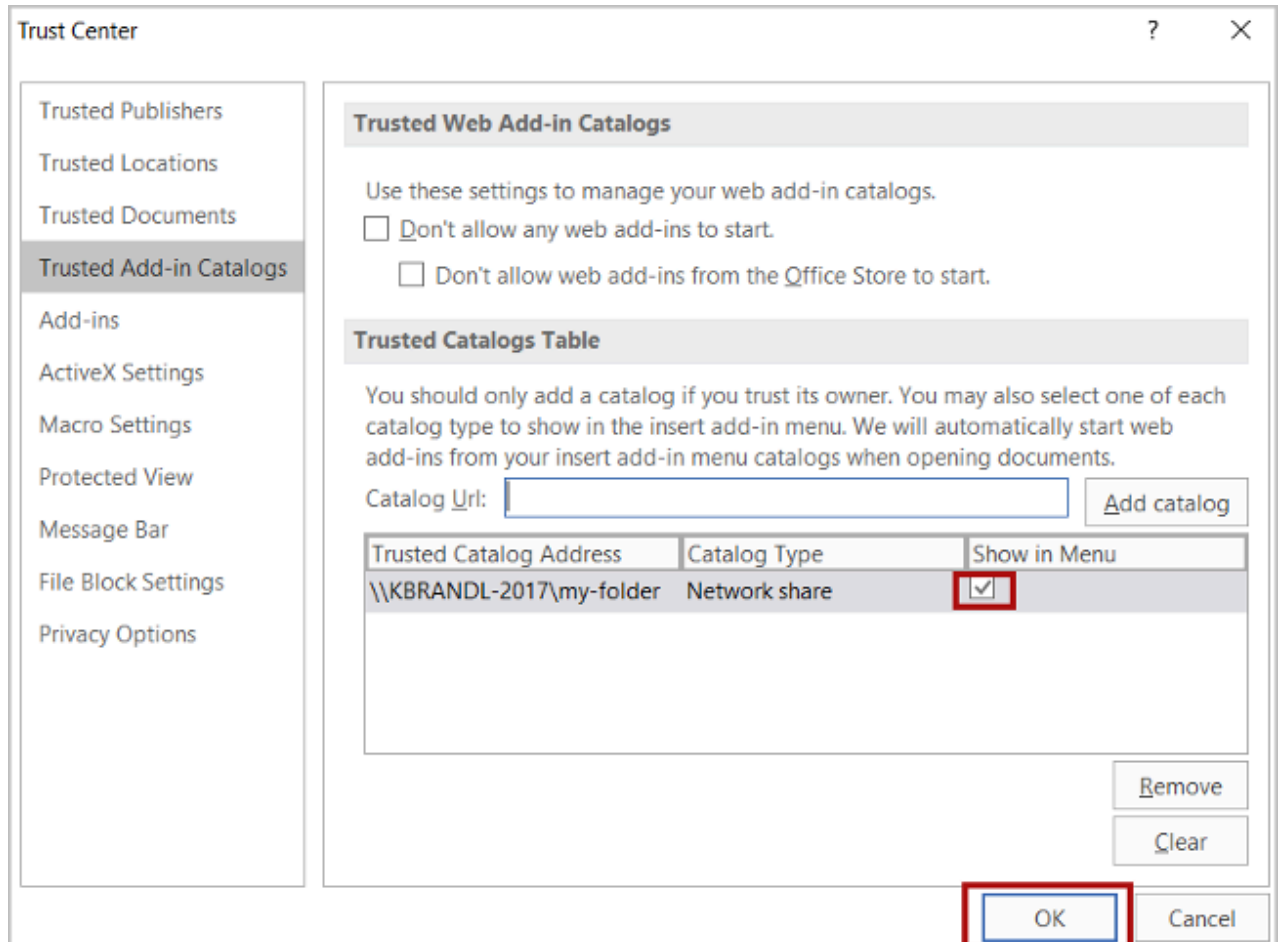
- [Configure the trust manually.](#)
- [Configure the trust with a Registry script.](#)

Configure the trust manually

1. Open a new document in Excel, Word, PowerPoint, or Project.
2. Choose the **File** tab, and then choose **Options**.
3. Choose **Trust Center**, and then choose the **Trust Center Settings** button.
4. Choose **Trusted Add-in Catalogs**.
5. In the **Catalog Url** box, enter the full network path to the folder that you [shared](#) previously. If you failed to note the folder's full network path when you shared the folder, you can get it from the folder's **Properties** dialog window, as shown in the following screenshot.



6. After you've entered the full network path of the folder into the **Catalog Url** box, choose the **Add catalog** button.
7. Select the **Show in Menu** check box for the newly-added item, and then choose the **OK** button to close the **Trust Center** dialog window.



8. Choose the **OK** button to close the **Options** dialog window.
9. Close and reopen the Office application so your changes will take effect.

Configure the trust with a Registry script

1. In a text editor, create a file named **TrustNetworkShareCatalog.reg**.
2. Add the following content to the file.

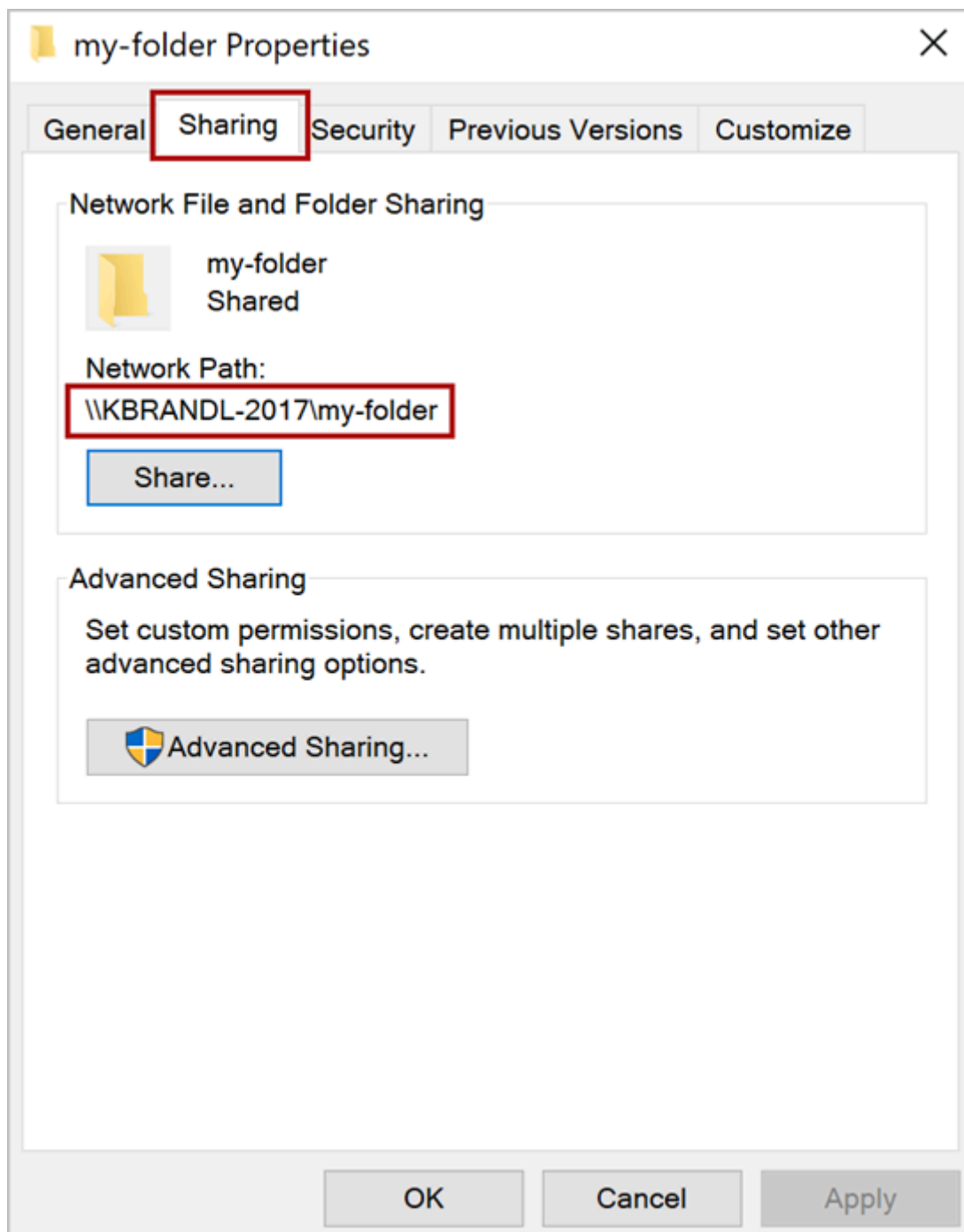
text

Windows Registry Editor Version 5.00

```
[HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\WEF\TrustedCatalogs\{-random-GUID-here-}]
"Id"="{-random-GUID-here-}"
```

```
"Url"="\-share-\-folder-"  
"Flags"=dword:00000001
```

3. Use one of the many online GUID generation tools, such as [GUID Generator](#), to generate a random GUID, and within the TrustNetworkShareCatalog.reg file, replace the string "-random-GUID-here-" *in both places* with the GUID. (The enclosing {} symbols should remain.)
4. Replace the `Url` value with the full network path to the folder that you [shared](#) previously. (Note that any \ characters in the URL must be doubled.) If you failed to note the folder's full network path when you shared the folder, you can get it from the folder's **Properties** dialog window, as shown in the following screenshot.



5. The file should now look like the following. Save it.

text

Windows Registry Editor Version 5.00

```
[HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\WEF\TrustedCatalogs\
{01234567-89ab-cedf-0123-456789abcdef}]
"Id"="{01234567-89ab-cedf-0123-456789abcdef}"
"Url"="\\\\TestServer\OfficeAddinManifests"
"Flags"=dword:00000001
```

6. Close *all* Office applications.
7. Run the TrustNetworkShareCatalog.reg just as you would any executable, such as double-clicking it.

Sideload your add-in

1. Put the manifest XML file of any add-in that you're testing into the shared folder catalog. Note that you deploy the web application itself to a web server. Be sure to specify the URL in the **<SourceLocation>** element of the manifest file.

Important

While not strictly required in all add-in scenarios, using an HTTPS endpoint for your add-in is strongly recommended. Add-ins that are not SSL-secured (HTTPS) generate unsecure content warnings and errors during use. If you plan to run your add-in in Office on the web or publish your add-in to AppSource, it must be SSL-secured. If your add-in accesses external data and services, it should be SSL-secured to protect data in transit. Self-signed certificates can be used for development and testing, so long as the certificate is trusted on the local machine.

Note

For Visual Studio projects, use the manifest built by the project in the `{projectfolder}\bin\Debug\OfficeAppManifests` folder.

2. In Excel, Word, or PowerPoint, select **Home > Add-ins** from the ribbon, then select **Advanced**. In Project, select **My Add-ins** on the **Project** tab of the ribbon.
3. Choose **SHARED FOLDER** at the top of the **Office Add-ins** dialog box.

4. Select the name of the add-in and choose **Add** to insert the add-in.

Remove a sideloaded add-in

You can remove a previously sideloaded add-in by clearing the Office cache on your computer. Details on how to clear the cache on Windows can be found in the article [Clear the Office cache](#).

See also

- [Validate an Office Add-in's manifest](#)
- [Clear the Office cache](#)
- [Publish your Office Add-in](#)

Attach a debugger from the task pane

Article • 05/20/2023

In some environments, a debugger can be attached on an Office Add-in that is already running. This can be useful when you want to debug an add-in that is already in staging or production. If you are still developing and testing the add-in, see [Overview of debugging Office Add-ins](#).

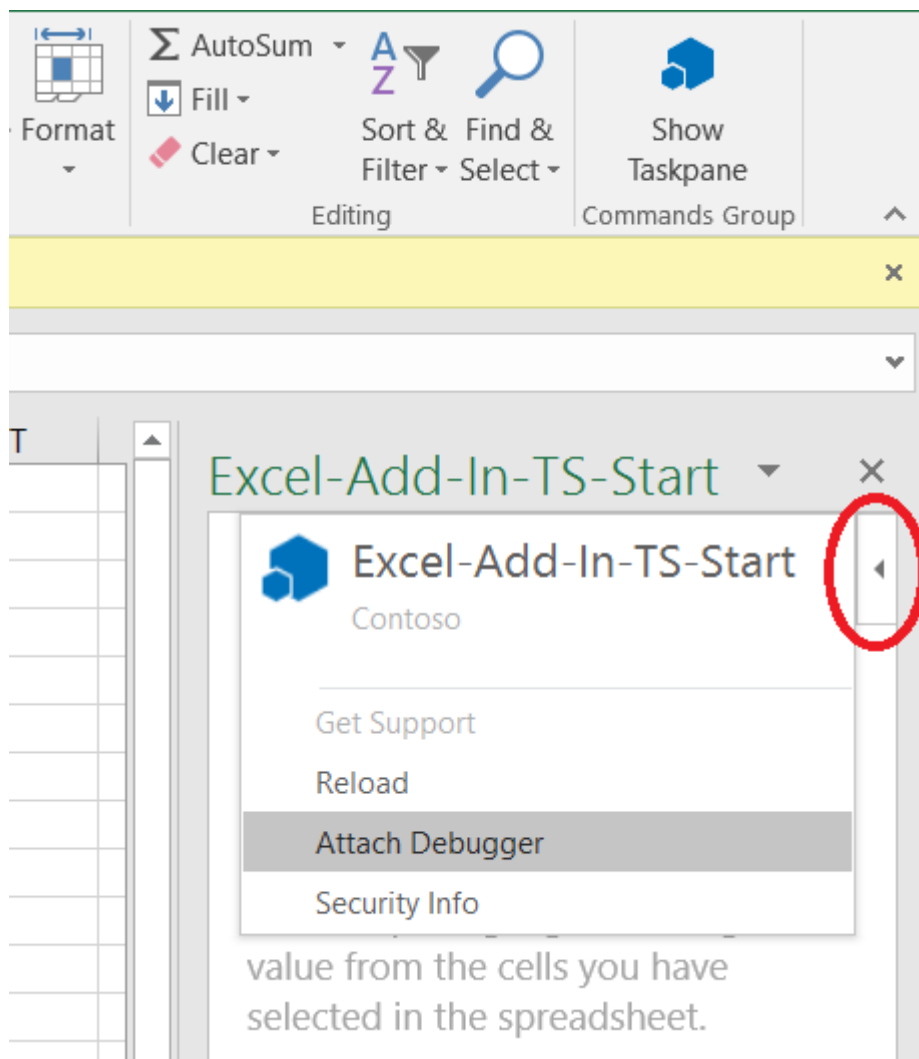
The technique described in this article can be used only when the following conditions are met.

- The add-in is running in Office on Windows.
- The computer is using a combination of Windows and Office versions that use the Edge (Chromium-based) webview control, WebView2. 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.

To launch the debugger, choose the top right corner of the task pane to activate the **Personality** menu (as shown in the red circle in the following image).



Select **Attach Debugger**. This launches the Microsoft Edge (Chromium-based) developer tools. Use the tools as described in [Debug add-ins using developer tools in Microsoft Edge \(Chromium-based\)](#).

See also

- [Overview of debugging Office Add-ins](#)