Import an add-in project to Microsoft 365 Agents Toolkit

Article • 05/19/2025

The Agents Toolkit extension for Visual Studio Code is a richly featured tool for working with extensions on the Microsoft 365 developer platform, including Teams apps, Office Add-ins, and Copilot extensions, among others. It also makes it easy to work with extensions that transcend the boundaries between Teams apps, add-ins, and Copilot extensions. For example, it makes sideloading such cross-boundary extensions easier.

There can be no algorithmic procedure for importing an add-in into the toolkit because an algorithm would have to make assumptions about the following aspects of the project.

- The folder and file structure of the existing add-in. But these structures vary depending on which tool was used to create the project and what version of that tool. The developer of the add-in might also have changed the structure after the project was created.
- The settings in various configuration files. But these settings also vary depending on how the project was created and changes made to the configuration since it was created.
- Which language, TypeScript or JavaScript, was used for the client-side source code of the web application.

However, we can make some general recommendations.

Note

- This article doesn't apply to add-in projects that were created with Visual Studio. Such projects are based on the ASP.NET web application framework and are designed to run on Internet Information Server (IIS). Converting such a project to work in Agents Toolkit would significantly more difficult and is out-of-scope for this article.
- Add-in projects in Agents Toolkit must use the unified manifest for Microsoft 365. If your add-in project uses a feature that isn't yet supported with the unified manifest, then you can't import it to Agents Toolkit unless you first redesign it so that it doesn't use unsupported features.
- Currently, add-ins that use the unified manifest can't be sideloaded on a Mac. If your development computer is a Mac, don't import your project into Agents Toolkit until sideloading on the Mac is supported.

There are two basic strategies available.

- Use the importation feature of the toolkit
- Start with a new toolkit project

Regardless of which you choose, begin by ensuring that you have installed Visual Studio Code and the Agents Toolkit extension.

Use the importation feature of the toolkit

There are four tasks to using the importation feature.

- Prepare the existing manifest
- Import the project
- Verify that the add-in can be sideloaded
- Post importation: Adjust the project structure as needed

Prepare the existing manifest

Important

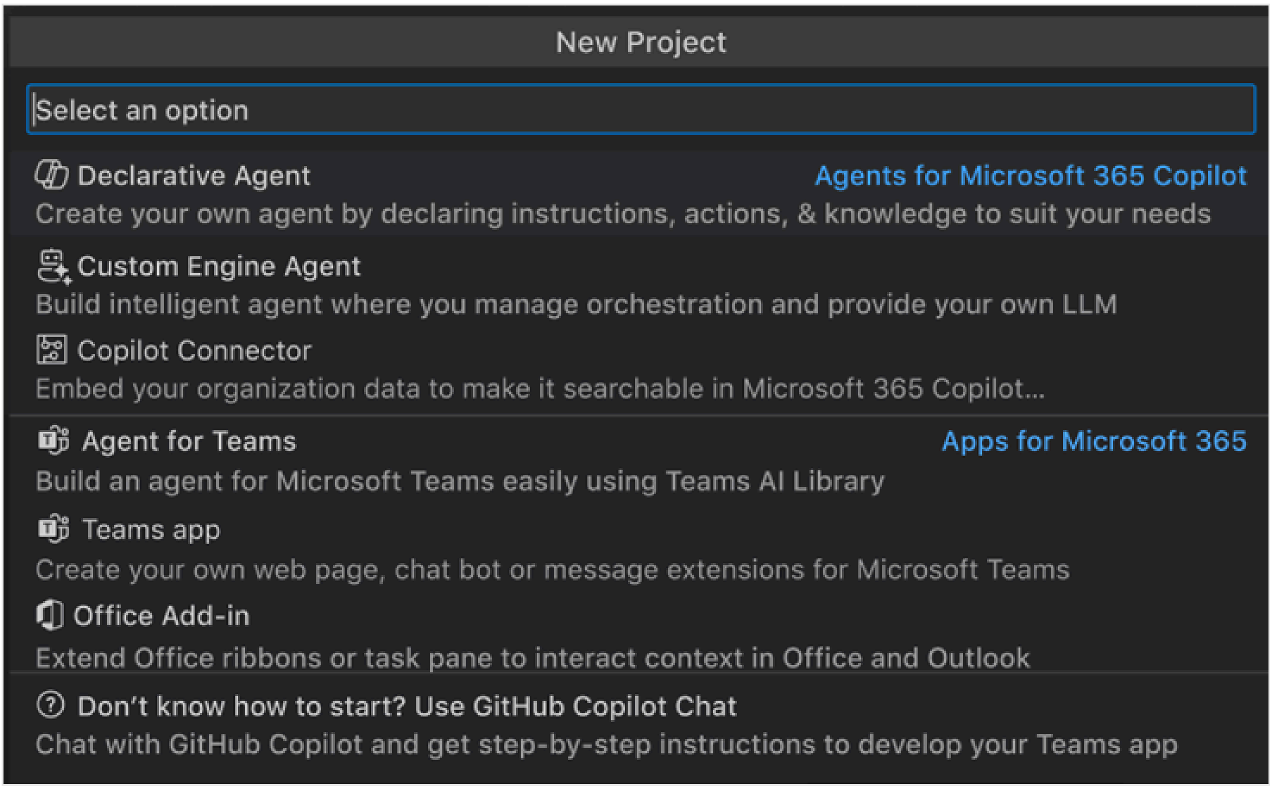
If the existing project uses the add-in only manifest, the importation feature automatically converts it to a unified manifest. So, you must carry out the steps in Ensure that your manifest is ready to convert before you import the project.

Import the project

1. Open Visual Studio Code and select the Agents Toolkit icon on the **Activity Bar**.

2. Select **Create a New App**.

3. The **New Project** dropdown menu opens. The options listed vary depending on your version of Agents Toolkit. Select **Office Add-in**.


New Project

Select an option

Declarative Agent — Agents for Microsoft 365 Copilot
Create your own agent by declaring instructions, actions, & knowledge to suit your needs

Custom Engine Agent
Build intelligent agent where you manage orchestration and provide your own LLM

Copilot Connector
Embed your organization data to make it searchable in Microsoft 365 Copilot...

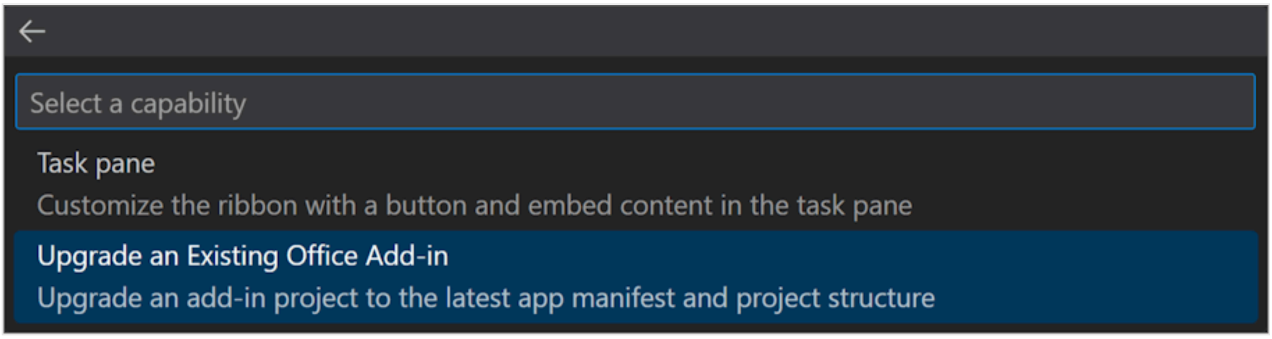
Agent for Teams — Apps for Microsoft 365
Build an agent for Microsoft Teams easily using Teams AI Library

Teams app
Create your own web page, chat bot or message extensions for Microsoft Teams

Office Add-in
Extend Office ribbons or task pane to interact context in Office and Outlook

Don't know how to start? Use GitHub Copilot Chat
Chat with GitHub Copilot and get step-by-step instructions to develop your Teams app

4. The **App Features Using an Office Add-in** dropdown menu opens. The options listed vary depending on your version of Agents Toolkit. Select **Upgrade an Existing Office Add-in**.


Select a capability

Task pane
Customize the ribbon with a button and embed content in the task pane

Upgrade an Existing Office Add-in
Upgrade an add-in project to the latest app manifest and project structure

5. In the **Existing add-in project folder** dropdown menu, browse to the root folder of the add-in project.

6. In the **Select import project manifest file** dropdown menu, browse to the add-in only manifest file, typically named **manifest.xml**.

7. In the **Workspace folder** dialog, select the folder where you want to put the converted project.

8. In the **Application name** dialog, give a name to the project (with no spaces). Agents Toolkit creates the project with your source files and scaffolding. It then opens the project *in a second Visual Studio Code window*. Close the original Visual Studio Code window.

Verify that the add-in can be sideloaded

ⓘ Note

Add-ins that use the unified manifest can be sideloaded only on Office Version 2304 (Build 16320.20000) or later.

Sideload the add-in using the instructions at Sideload with Microsoft 365 Agents Toolkit.

If you encounter problems, as a troubleshooting step, try sideloading with a system prompt, bash shell, or terminal. If you can, then the problem is isolated to the toolkit.

It's possible that sideloading problems are the result of a file and folder structure, or configuration settings, that are different from what Agents Toolkit normally expects. See the section Post importation: Adjust the project structure and settings as needed.

Post importation: Adjust the project structure and settings as needed

The importation process creates some folders and files that Visual Studio Code or Agents Toolkit need, but it doesn't reorganize your source files; such as HTML, JavaScript, and CSS files. It also doesn't change the content of any files in the project, including tool configuration files. We recommend that you change your project to match the pattern of projects that are created in Agents Toolkit. As you work, keep the following points in mind.

- HTML files in the toolkit projects don't have inline `<script>` elements. They only use `<script>` elements with a `src` attribute that loads an external file.
- Source files in a new toolkit project are in a folder named **\src**. Within this folder, the source files are further divided into subfolders based on the runtimes in which they normally are run. The following is a typical structure.

  Console

  ```
  \src
      \commands
         commands.html
         commands.js
      \taskpane
         taskpane.css
         taskpane.html
         taskpane.js
  ```

- Agents Toolkit projects have a folder named **\appPackage**. The manifest and any other files that should be in the app package zip file are in this folder.

ⓘ **Important**

- The URLs in manifest will reflect the original structure of the project. Change these URLs as needed if you make changes in the file and folder structure.
- Tool configuration files, such as webpack.config.js, may have URLs. Change these as needed.

Start with a new toolkit project

As an alternative to using the toolkit's importation feature, you can create a brand new add-in project in the toolkit and move files from the existing project into it and make changes to other files. The following are the tasks that you need to carry out.

1. If the existing project uses the add-in only manifest, convert it. See Convert an add-in to use the unified manifest for Microsoft 365.
2. Create a new add-in project in Agents Toolkit. For each choice the toolkit asks you to make, such as the choice between JavaScript and TypeScript, make the choice that best matches your existing add-in. See Create Office Add-in projects with Microsoft 365 Agents Toolkit.
3. Replace the manifest in the new project's **\appPackage** folder with your converted manifest.

   ⓘ **Note**

   If the conversion process produced any language string files, such as **fr-fr.json**, add these to the **\appPackage** folder.

4. Replace the files in the **\src** folder of the new project with the source files from your old project. To maximize compatibility with the configuration files in the new project, we recommend that you divide your source files into subfolders based on the runtimes in which they normally are run. For example, have separate folders for the source files of function commands, the taskpane, autorun events, and Excel custom functions.
5. Edit the manifest to ensure that any URLs in it are compatible with the new structure of the project.
6. Inspect the configuration files in the new project to ensure that they are compatible with the organization of the project.

> **Tip**
>
> When the old project and the new both have a configuration file with the same name (such as **babel.config.json**), use a file comparison ("diff") tool to find the differences. For each difference, determine which file is correct for the new project and change the file in the new project as needed.

7. The **webpack.config.js** is likely to need editing. It isn't possible to give universal rules for that file, but the following principles apply in most cases.

   - Ensure that URLs match the structure of the project.
   - Ensure that there is an `entry` subproperty for each subfolder under the **\src** folder.
   - Ensure that the `plugins` array also takes account of each subfolder under the **\src** folder.
   - Ensure that the `extensions` and `rules` properties take account of the file types in your project that should be handled by loaders and bundled.

8. Ensure that you can sideload the add-in in the new project. See Verify that the add-in can be sideloaded.