Build an Excel task pane add-in with Visual Studio

Article • 08/27/2024

In this article, you'll walk through the process of building an Excel task pane add-in in Visual Studio.

Prerequisites

- Visual Studio 2019 or later with the **Office/SharePoint development** workload installed.

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

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

Create the add-in project

1. In Visual Studio, choose **Create a new project**.
2. Using the search box, enter **add-in**. Choose **Excel Web Add-in**, then select **Next**.
3. Name your project **ExcelWebAddIn1** and select **Create**.
4. In the **Create Office Add-in** dialog window, choose **Add new functionalities to Excel**, and then choose **Finish** to create the project.
5. Visual Studio creates a solution and its two projects appear in **Solution Explorer**. The **Home.html** file opens in Visual Studio.

Explore the Visual Studio solution

When you've completed the wizard, Visual Studio creates a solution that contains two projects.

Expand table

Project	Description
Add-in project	Contains only an XML-formatted add-in only manifest file, which contains all the settings that describe your add-in. These settings help the Office application determine when your add-in should be activated and where the add-in should appear. Visual Studio generates the contents of this file for you so that you can run the project and use your add-in immediately. Change these settings any time by modifying the XML file.
Web application project	Contains the content pages of your add-in, including all the files and file references that you need to develop Office-aware HTML and JavaScript pages. While you develop your add-in, Visual Studio hosts the web application on your local IIS server. When you're ready to publish the add-in, you'll need to deploy this web application project to a web server.

Update the code

1. **Home.html** specifies the HTML that will be rendered in the add-in's task pane. In **Home.html**, replace the `<body>` element with the following markup and save the file.

HTML

```html
<body class="ms-font-m ms-welcome">
    <div id="content-header">
        <div class="padding">
            <h1>Welcome</h1>
        </div>
    </div>
    <div id="content-main">
        <div class="padding">
            <p>Choose the button below to set the color of the selected 
range to green.</p>
            <br />
            <h3>Try it out</h3>
            <button class="ms-Button" id="set-color">Set color</button>
        </div>
    </div>
</body>
```

2. Open the file **Home.js** in the root of the web application project. This file specifies the script for the add-in. Replace the entire contents with the following code and save the file.

JavaScript

```
'use strict';

(function () {

    Office.onReady(function() {
        // Office is ready.
        $(document).ready(function () {
            // The document is ready.
            $('#set-color').on("click", setColor);
        });
    });

    async function setColor() {
        await Excel.run(async (context) => {
            const range = context.workbook.getSelectedRange();
            range.format.fill.color = 'green';

            await context.sync();
        }).catch(function (error) {
            console.log("Error: " + error);
            if (error instanceof OfficeExtension.Error) {
                console.log("Debug info: " +
JSON.stringify(error.debugInfo));
            }
        });
    }
})();
```

3. Open the file **Home.css** in the root of the web application project. This file specifies the custom styles for the add-in. Replace the entire contents with the following code and save the file.

```css
#content-header {
    background: #2a8dd4;
    color: #fff;
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 80px;
    overflow: hidden;
}

#content-main {
    background: #fff;
    position: fixed;
    top: 80px;
    left: 0;
    right: 0;
    bottom: 0;
    overflow: auto;
}

.padding {
    padding: 15px;
}
```

Update the manifest

1. In **Solution Explorer**, go to the **ExcelWebAddIn1** add-in project and open the **ExcelWebAddIn1Manifest** directory. This directory contains your manifest file, **ExcelWebAddIn1.xml**. The manifest file defines the add-in's settings and capabilities. See the preceding section Explore the Visual Studio solution for more information about the two projects created by your Visual Studio solution.

2. The `ProviderName` element has a placeholder value. Replace it with your name.

3. The `DefaultValue` attribute of the `DisplayName` element has a placeholder. Replace it with **My Office Add-in**.

4. The `DefaultValue` attribute of the `Description` element has a placeholder. Replace it with **A task pane add-in for Excel**.

5. Save the file.

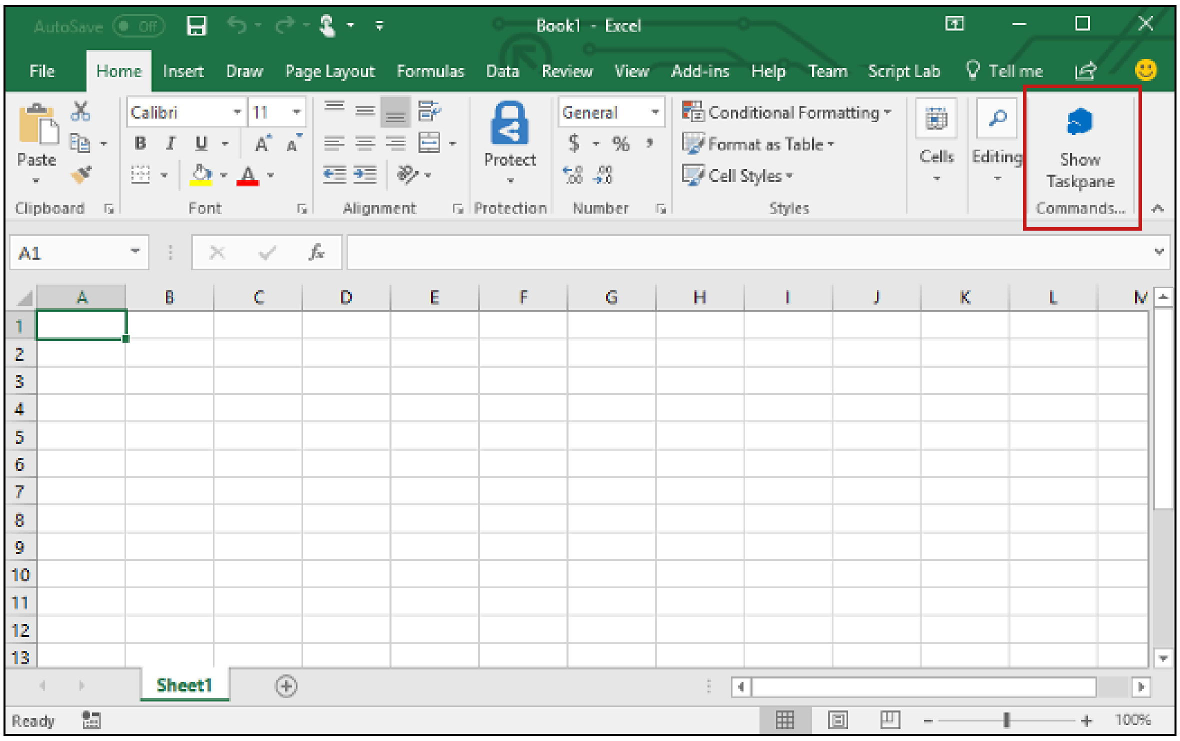
   XML

   ```xml
   ...
   <ProviderName>John Doe</ProviderName>
   <DefaultLocale>en-US</DefaultLocale>
   <!-- The display name of your add-in. Used on the store and various
   places of the Office UI such as the add-ins dialog. -->
   <DisplayName DefaultValue="My Office Add-in" />
   <Description DefaultValue="A task pane add-in for Excel"/>
   ...
   ```

Try it out

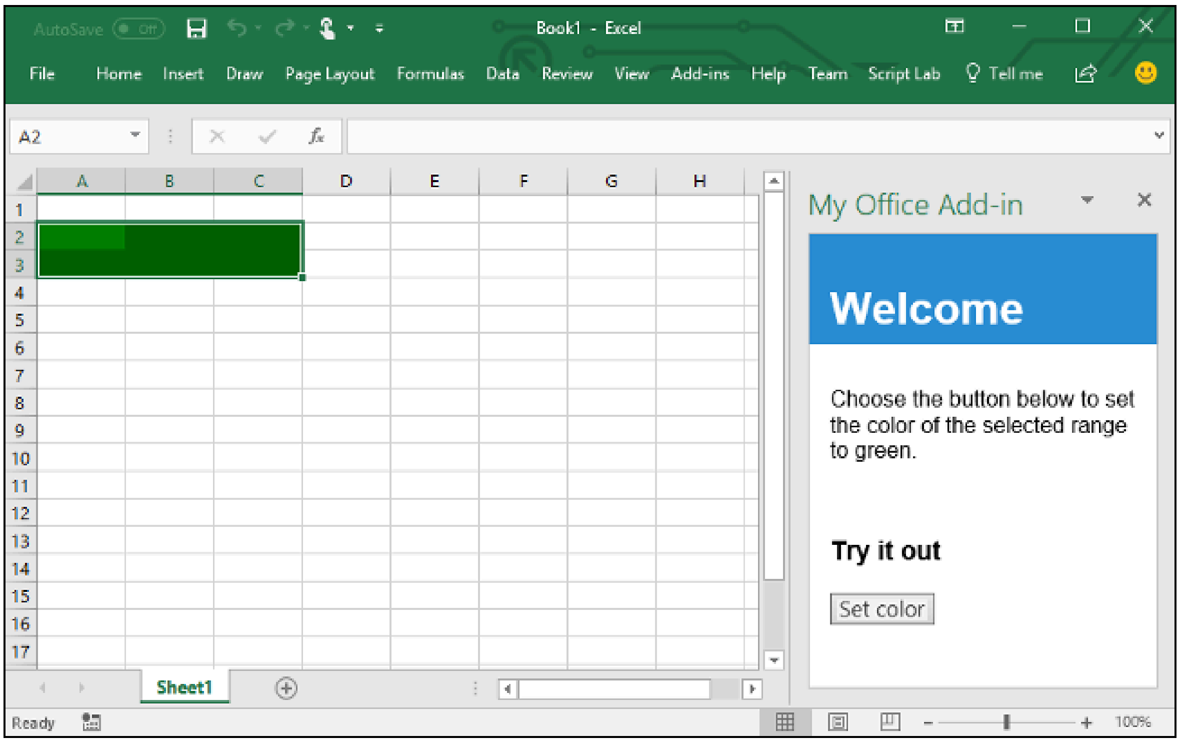
1. Using Visual Studio, test the newly created Excel add-in by pressing `F5` or choosing the **Start** button to launch Excel with the **Show Taskpane** add-in button displayed on the ribbon. The add-in will be hosted locally on IIS. If you're asked to trust a certificate, do so to allow the add-in to connect to its Office application.

2. In Excel, choose the **Home** tab, and then choose the **Show Taskpane** button on the ribbon to open the add-in task pane.



3. Select any range of cells in the worksheet.

4. In the task pane, choose the **Set color** button to set the color of the selected range to green.



Choose the button below to set the color of the selected range to green.

ⓘ Note

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

Next steps

Congratulations, you've successfully created an Excel task pane add-in! Next, learn more about developing Office Add-ins with Visual Studio.

Troubleshooting

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

Code samples

- Excel "Hello world" add-in: Learn how to build a simple Office Add-in with only a manifest, HTML web page, and a logo.

See also

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

- Excel JavaScript object model in Office Add-ins
- Excel add-in code samples
- Excel JavaScript API reference
- Publish your add-in using Visual Studio

Build an Excel content add-in

Article • 08/27/2024

In this article, you'll walk through the process of building an Excel content add-in using Visual Studio.

Prerequisites

- Visual Studio 2019 or later with the **Office/SharePoint development** workload installed.

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

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

Create the add-in project

1. In Visual Studio, choose **Create a new project**.
2. Using the search box, enter **add-in**. Choose **Excel Web Add-in**, then select **Next**.
3. Name your project **ExcelWebAddIn1** and select **Create**.
4. In the **Create Office Add-in** dialog window, choose the **Insert content into Excel spreadsheets** add-in type, then choose **Next**.
5. Choose the **Basic Add-in** or **Document Visualization Add-in** add-in template, and then choose **Finish** to create the project.
6. Visual Studio creates a solution and its two projects appear in **Solution Explorer**. The **Home.html** file opens in Visual Studio.

Explore the Visual Studio solution

When you've completed the wizard, Visual Studio creates a solution that contains two projects.

Expand table

Project	Description
Add-in project	Contains only an XML-formatted add-in only manifest file, which contains all the settings that describe your add-in. These settings help the Office application determine when your add-in should be activated and where the add-in should appear. Visual Studio generates the contents of this file for you so that you can run the project and use your add-in immediately. Change these settings any time by modifying the XML file.
Web application project	Contains the content pages of your add-in, including all the files and file references that you need to develop Office-aware HTML and JavaScript pages. While you develop your add-in, Visual Studio hosts the web application on your local IIS server. When you're ready to publish the add-in, you'll need to deploy this web application project to a web server.

Update the manifest

1. In **Solution Explorer**, go to the **ExcelWebAddIn1** add-in project and open the **ExcelWebAddIn1Manifest** directory. This directory contains your manifest file, **ExcelWebAddIn1.xml**. The manifest file defines the add-in's settings and capabilities. See the preceding section Explore the Visual Studio solution for more information about the two projects created by your Visual Studio solution.

2. The `ProviderName` element has a placeholder value. Replace it with your name.

3. The `DefaultValue` attribute of the `DisplayName` element has a placeholder. Replace it with **My Office Add-in**.

4. The `DefaultValue` attribute of the `Description` element has a placeholder. Replace it with **A content add-in for Excel.**.

5. Save the file. The updated lines should look like the following code sample.

XML

```xml
...
<ProviderName>John Doe</ProviderName>
<DefaultLocale>en-US</DefaultLocale>
<!-- The display name of your add-in. Used on the store and various
places of the Office UI such as the add-ins dialog. -->
<DisplayName DefaultValue="My Office Add-in" />
<Description DefaultValue="A content add-in for Excel."/>
...
```

Try it out

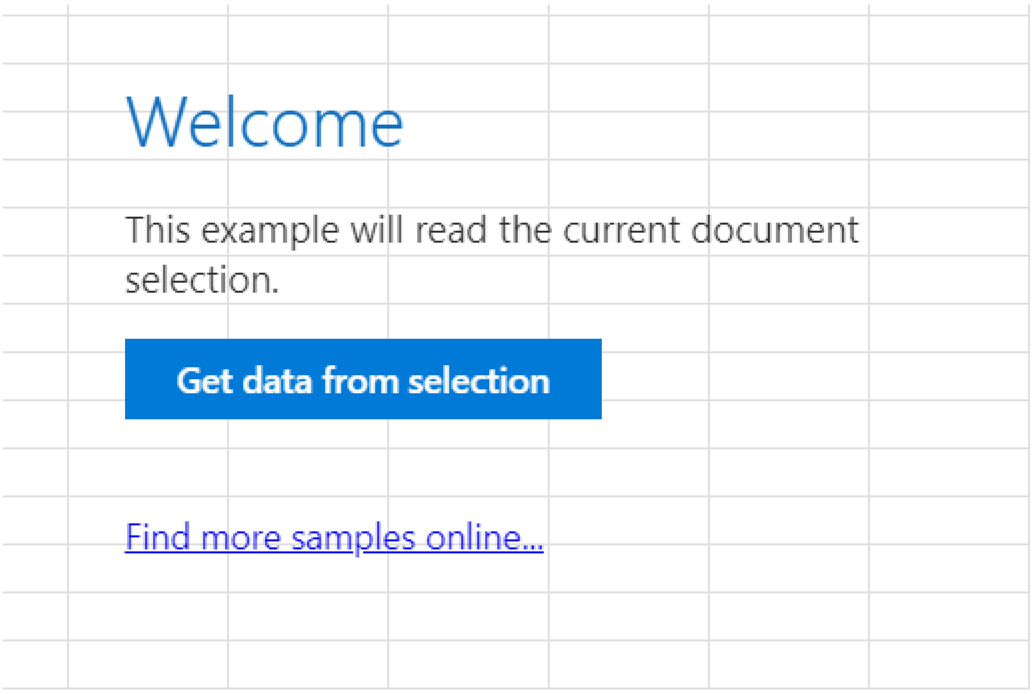
1. Using Visual Studio, test the newly created Excel add-in by pressing `F5` or choosing the **Start** button to launch Excel with the content add-in displayed in the spreadsheet.

2. Ensure that there's text in the worksheet, then select any range of cells containing text in the worksheet.

3. Select the tab for the template you chose, then follow the instructions.

Basic Add-in

- In the content add-in, choose the **Get data from selection** button to get the text from the selected range.



Welcome

This example will read the current document selection.

ⓘ Note

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

Next steps

Congratulations, you've successfully created an Excel content add-in! Next, learn more about developing Office Add-ins with Visual Studio.

Troubleshooting

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

Code samples

- Excel Content Add-in Humongous Insurance

See also

- Office Add-ins platform overview
- Develop Office Add-ins
- Excel JavaScript object model in Office Add-ins
- Excel add-in code samples
- Excel JavaScript API reference
- Using Visual Studio Code to publish