

# Runtimes in Office Add-ins

07/03/2025

Office Add-ins execute in runtimes embedded in Office. As an interpreted language, JavaScript must run in a JavaScript runtime. [Node.js](#) and modern browsers are examples of such runtimes.


## Types of runtimes

There are two types of runtimes used by Office Add-ins:

- **JavaScript-only runtime:** A JavaScript engine supplemented with support for [WebSockets](#), [Full CORS \(Cross-Origin Resource Sharing\)](#), and client-side storage of data. It doesn't support [local storage](#) or cookies.
- **Browser runtime:** Includes all the features of a JavaScript-only runtime and adds support for [local storage](#), a [rendering engine](#) that renders HTML, and cookies.

Details about these types are later in this article at [JavaScript-only runtime](#) and [Browser runtime](#).

The following table shows the possible features of an add-in, according to each type of runtime.

 Expand table

| Type of runtime | Add-in feature  |
|-----------------|---|
| JavaScript-only | Excel <a href="#">custom functions</a><br>(except when the runtime is <a href="#">shared</a> or the add-in is running in Office on the web) |
|                 | <a href="#">Excel, PowerPoint, or Word event-based task</a><br>(only when the add-in is running on Windows)                                 |
|                 | <a href="#">Outlook event-based task</a><br>(only when the add-in is running in classic Outlook on Windows)                                 |
|                 | <a href="#">Outlook integrated spam reporting feature</a><br>(only when the add-in is running in classic Outlook on Windows)                |
| browser         | <a href="#">task pane</a>   |
|                 | <a href="#">dialog</a>  |
|                 | <a href="#">function command</a>  |

| Type of runtime | Add-in feature  |
|-----------------|---|
|                 | <p>Excel <a href="#">custom functions</a><br/>(when the runtime is <a href="#">shared</a> or the add-in is running in Office on the web)</p> <p><a href="#">Excel, PowerPoint, or Word event-based task</a><br/>(only when the add-in is running on Mac or the web)</p> <p><a href="#">Outlook event-based task</a><br/>(when the add-in is running in Outlook on Mac or Outlook on the web or in the <a href="#">new Outlook on Windows</a> <a href="#">↗</a>)</p> <p><a href="#">Outlook integrated spam reporting feature</a><br/>(only when the add-in is running in Outlook on Mac or on the web or in the <a href="#">new Outlook on Windows</a> <a href="#">↗</a>)</p> |

The following table shows the same information organized by which type of runtime is used for the various possible features of an add-in.

[↗](#) Expand table

| Add-in feature                            | Type of runtime on Windows   | Type of runtime on Mac | Type of runtime on the web |
|---|--|------------------------|----------------------------|
| Excel custom functions                    | JavaScript-only<br>(but <i>browser</i> when the runtime is shared)                         | browser                | browser                    |
| Outlook event-based tasks                 | JavaScript-only<br>(classic Outlook on Windows)<br><br>browser<br>(new Outlook on Windows) | browser                | browser                    |
| Event-based tasks on other applications   | JavaScript-only  | browser                | browser                    |
| Outlook integrated spam reporting feature | JavaScript-only<br>(classic Outlook on Windows)<br><br>browser<br>(new Outlook on Windows) | browser                | browser                    |
| task pane                                 | browser  | browser                | browser                    |

| Add-in feature   | Type of runtime on Windows | Type of runtime on Mac | Type of runtime on the web |
|------------------|----------------------------|------------------------|----------------------------|
| dialog           | browser                    | browser                | browser                    |
| function command | browser                    | browser                | browser                    |

In Office on the web, everything always runs in a browser type runtime. With one exception, everything in an add-in on the web runs in the *same* browser process: the browser process in which the user has opened Office on the web. The exception is when a dialog is opened with a call of [Office.ui.displayDialogAsync](#) and the [DialogOptions.displayInIFrame](#) option is *not* passed and set to `true`. When the option isn't passed (so it has the default `false` value), the dialog opens in its own process. The same principle applies to the [OfficeRuntime.displayWebDialog](#) method and the [OfficeRuntime.DisplayWebDialogOptions.displayInIFrame](#) option.

When an add-in is running on a platform other than the web, the following principles apply.

- A dialog runs in its own runtime process.
- An Outlook event-based or spam-reporting add-in runs in its own runtime process.

#### ⓘ Note

The [event-based activation](#) and [integrated spam reporting](#) features in Outlook must use the same runtime. Multiple runtimes aren't currently supported in Outlook.

- By default, task panes, function commands, and Excel custom functions each run in their own runtime process. However, for some Office host applications, the add-in manifest can be configured so that any two, or all three, can run in the same runtime. See [Shared runtime](#).

Depending on the host Office application and the features used in the add-in, there may be many runtimes in an add-in. Each usually will run in its own process but not necessarily simultaneously. The following are examples.

- A PowerPoint or Word add-in that doesn't share any runtimes, and includes the following features, has as many as three runtimes.
  - A task pane
  - A function command
  - A dialog (A dialog can be launched from either the task pane or the function command.)

### ⓘ Note

It's not a good practice to have multiple dialogs open simultaneously, but if the add-in enables the user to open one from the task pane and another from the function command at the same time, this add-in would have four runtimes. A task pane, and a given invocation of a function command can have only one open dialog at a time; but if the function command is invoked multiple times, a new dialog is opened on top of its predecessor with each invocation, so there could be many runtimes. The remainder of this list ignores the possibility of multiple open dialogs.

- An Excel add-in that doesn't share any runtimes, and includes the following features, has as many as *four* runtimes.
  - A task pane
  - A function command
  - A custom function
  - A dialog (A dialog can be launched from either the task pane, the function command, or a custom function.)
- An Excel add-in with the same features and is configured to share the same runtime across the task pane, function command, and custom function, has *two* runtimes. A shared runtime can open only one dialog at a time.
- An Excel add-in with the same features, except that it has no dialog, and is configured to share the same runtime across the task pane, function command, and custom function, has *one* runtime.
- An Outlook add-in that has the following features has as many as *four* runtimes. Shared runtimes aren't supported in Outlook.
  - A task pane
  - A function command
  - An event-based task or integrated spam reporting feature
  - A dialog (A dialog can be launched from either the task pane or the function command, but not from an event-based task.)

## Share data across runtimes

### ⓘ Note

- If you know that your add-in will only be used in Office on the web and that it won't open any dialogs with the `displayInIFrame` option set to `true`, then you can ignore this section. Since everything in your add-in runs in the same runtime process, you can just use global variables to share data between features.
- As noted above in [Types of runtimes](#), the type of runtime used by a feature varies partly by platform. It's a good practice to avoid having add-in code that branches based on platform, so the guidance in this section recommends techniques that will work cross-platform. There is only one case, noted below, in which branching code is required.

For Excel, PowerPoint, and Word add-ins, use a [Shared runtime](#) when any two or more features, except dialogs, need to share data. In Outlook, or scenarios where sharing a runtime isn't feasible, you need alternative methods. The parts of the add-in that are in separate runtime processes don't share global data automatically and are treated by the add-in's web application server as separate sessions, so [Window.sessionStorage](#) can't be used to share data between them. *The following guidance assumes that you're not using a shared runtime.*

- Pass data between a dialog and its parent task pane, function command, or custom function by using the [Office.ui.messageParent](#) and [Dialog.messageChild](#) methods.

ⓘ **Note**

The `OfficeRuntime.storage` methods can't be called in a dialog, so this isn't an option for sharing data between a dialog and another runtime.

- To share data between a task pane and a function command, store data in [Window.localStorage](#), which is shared across all runtimes that access the same specific origin.

ⓘ **Note**

LocalStorage isn't accessible in a JavaScript-only runtime and, thus, it isn't available in Excel custom functions. It also can't be used to share data with an Outlook event-based tasks (since those tasks use a JavaScript-only runtime on some platforms).

Starting in Version 115 of Chromium-based browsers, such as Chrome and Edge, [storage partitioning](#) is enabled to prevent specific side-channel cross-site tracking (see also [Microsoft Edge browser policies](#)). This means that data stored by storage

APIs, such as local storage, are only available to contexts with the same origin and the same top-level site.

### 💡 Tip

Data in `Window.localStorage` persists between sessions of the add-in and is shared by add-ins with the same origin. Both of these characteristics are often undesirable for an add-in.

- To ensure that each session of a given add-in starts fresh call the [Window.localStorage.clear](#) method when the add-in starts.
- To allow some stored values to persist, but reinitialize other values, use [Window.localStorage.setItem](#) when the add-in starts for each item that should be reset to an initial value.
- To delete an item entirely, call [Window.localStorage.removeItem](#).

- To share data between an Excel custom function and any other runtime, use [OfficeRuntime.storage](#).
- To share data between an Outlook event-based task and a task pane or function command, you must branch your code by the value of the [Office.context.platform](#) property.
  - When the value is `PC` (Windows), store and retrieve data using the [Office.sessionData](#) APIs.
  - When the value is `Mac`, use `Window.localStorage` as described earlier in this list.

Other ways to share data include the following:

- Store shared data in an online database that is accessible to all the runtimes.
- Store shared data in a cookie for the add-in's domain to share it between browser runtimes. JavaScript-only runtimes don't support cookies.

For more information, see [Persist add-in state and settings](#) and [Get and set add-in metadata for an Outlook add-in](#).

## JavaScript-only runtime

The JavaScript-only runtime that is used in Office Add-ins is a modification of an open source runtime originally created for [React Native](#). It contains a JavaScript engine supplemented with support for [WebSockets](#), [Full CORS \(Cross-Origin Resource Sharing\)](#), and

[OfficeRuntime.storage](#). It doesn't have a rendering engine, and it doesn't support cookies or [local storage](#) <sup>↗</sup>.

This type of runtime is used in event-based and spam-reporting add-ins in classic Outlook on Windows only and in Excel custom functions *except* when the custom functions are [sharing a runtime](#).

- When used for an Excel custom function, the runtime starts up when either the worksheet recalculates or the custom function calculates. It doesn't shut down until the workbook is closed.
- When used in an Outlook event-based or spam-reporting add-in, the runtime starts up when the event occurs. It ends when the first of the following occurs.
  - The event handler calls the `completed` method of its event parameter.
  - Five minutes have elapsed since the triggering event.
  - The user changes focus from the window where the event was triggered, such as a message compose window (only applies to event-based add-ins).

#### ⓘ Note

The [event-based activation](#) and [integrated spam reporting](#) features in Outlook must use the same runtime. Multiple runtimes aren't currently supported in Outlook.

A JavaScript-only runtime uses less memory and starts up faster than a browser runtime, but has fewer features.

#### ⓘ Important

In Office for Windows versions before 2403 (Build 16.0.17425.20000) and perpetual license Office through Office 2021, the JavaScript-only runtime directly supports the ECMAScript 2016 standard of JavaScript. However, you can use later versions of JavaScript or TypeScript. For information about how to do this, see [Support for recent versions of JavaScript](#).

## Browser runtime

Office Add-ins use a different browser type runtime depending on the platform in which Office is running (web, Mac, or Windows), and on the version and build of Windows and Office. For example, if the user is running Office on the web in a Firefox browser, then the Firefox runtime is used. If the user is running Office on Mac, then the Safari runtime is used. If the user is

running Office on Windows, then either an Edge or Internet Explorer provides the runtime, depending on the version of Windows and Office. Details can be found in [Browsers and webview controls used by Office Add-ins](#).

All of these runtimes include an HTML rendering engine and provide support for [WebSockets](#), [Full CORS \(Cross-Origin Resource Sharing\)](#), and [local storage](#), and cookies.

A browser runtime lifespan varies depending on the feature that it implements and on whether it's being shared or not.

- When an add-in with a task pane is launched, a browser runtime starts, unless it's a shared runtime that is already running. If it's a shared runtime, it shuts down when the document is closed. If it isn't a shared runtime, it shuts down when the task pane is closed.
- When a dialog is opened, a browser runtime starts. It shuts down when the dialog is closed.
- When a function command is executed (which happens when a user selects its button or menu item), a browser runtime starts, unless it's a shared runtime that is already running. If it's a shared runtime, it shuts down when the document is closed. If it isn't a shared runtime, it shuts down when the first of the following occurs.
  - The function command calls the `completed` method of its event parameter.
  - Five minutes have elapsed since the triggering event. (If a dialog was opened in the function command and it's still open when the parent runtime times out, the dialog runtime stays running until the dialog is closed.)
- When an Excel custom function is using a shared runtime, then a browser-type runtime starts when the custom function calculates if the shared runtime hasn't already started for some other reason. It shuts down when the document is closed.

#### ⓘ Note

When a runtime is being shared, it's possible for your code to close the task pane without shutting down the add-in. See [Show or hide the task pane of your Office Add-in](#) for more information.

A browser runtime has more features than a JavaScript-only runtime, but starts up slower and uses more memory.

## Shared runtime



A "shared runtime" isn't a type of runtime. It refers to a [browser-type runtime](#) that's being shared by features of the add-in that would otherwise each have their own runtime. Specifically, you have the option of configuring the add-in's task pane and function commands to share a runtime. In an Excel add-in, you can also configure custom functions to share the runtime of a task pane or function command or both. When you do this, the custom functions are running in a browser-type runtime, instead of a [JavaScript-only runtime](#) as it otherwise would. See [Configure your add-in to use a shared runtime](#) for information about the benefits and limitations of sharing runtimes and instructions for configuring the add-in to use a shared runtime. In brief, the JavaScript-only runtime uses less memory and starts up faster, but has fewer features.

#### ⓘ Note

- You can share runtimes only in Excel, PowerPoint, and Word.
- You can't configure a dialog to share a runtime. Each dialog always has its own, except when the dialog is launched in Office on the web with the `displayInIFrame` option set to `true`.
- A shared runtime never uses the original Microsoft Edge WebView (EdgeHTML) runtime. If the conditions for using Microsoft Edge with WebView2 (Chromium-based) are met (as specified in [Browsers and webview controls used by Office Add-ins](#)), then that runtime is used. Otherwise, the Internet Explorer 11 runtime is used.