

# Resource limits and performance optimization for Office Add-ins

09/03/2025

Quality add-ins must perform within specific requirements for CPU core usage, memory usage, reliability, and, for Outlook add-ins, regular expression evaluation response time. These limits help ensure performance for your users and mitigate denial-of-service attacks. Be sure to test your Office Add-in on your target Office application by using a range of possible data, and measure its performance against the following run-time usage limits.

## Resource usage limits for add-ins

### ⓘ Note

The resource limits in this section only apply to Excel, Outlook on Mac (classic), PowerPoint, and Word.

The following runtime resource limits apply to add-ins running in Office clients on Windows and Mac, but not on mobile apps or in a browser.

- **CPU core usage** - A single CPU core usage threshold of 90%, observed three times in five-second intervals by default.

If the Office client detects the CPU core usage of an add-in is above the threshold value, it displays a message asking if the user wants to continue running the add-in. If the user chooses to continue, the Office client does not ask the user again during that edit session. The default interval for an Office client to check CPU core usage is every five seconds. Administrators can use the **AlertInterval** registry key to raise the threshold to reduce the display of this warning message if users run CPU-intensive add-ins.

- **Memory usage** - A default memory usage threshold that is dynamically determined based on the available physical memory of the device.

By default, when a Office client detects that physical memory usage on a device exceeds 80% of the available memory, the client starts monitoring the add-in's memory usage. This is done at the document level for content and task pane add-ins and at the mailbox level for Outlook add-ins. At a default interval of five seconds, the client warns the user if physical memory usage for a set of add-ins at the document or mailbox level exceeds 50%. This memory usage limit uses physical rather than virtual memory to ensure performance on devices with limited RAM, such as tablets. Administrators can override

this dynamic setting with an explicit limit by using the **MemoryAlertThreshold** Windows registry key as a global setting. They can also adjust the alert interval with the **AlertInterval** key.

- **Crash tolerance** - A default limit of four crashes during the document's session.

Administrators can adjust the threshold for crashes by using the **RestartManagerRetryLimit** registry key.

- **Application blocking** - A prolonged unresponsiveness threshold of five seconds.

This affects the user's experiences of the add-in and the Office application. When this occurs, the Office application automatically restarts all the active add-ins for a document or mailbox (where applicable), and warns the user which add-in became unresponsive. Add-ins reach this threshold when they don't regularly yield processing while performing long-running tasks. There are techniques listed later in this article to help ensure the add-in doesn't block the Office application. Administrators cannot override this threshold.

#### **Note**

Although only Outlook on Mac (classic) monitors resource usage, if the client makes an Outlook add-in unavailable, the add-in also become unavailable in other supported Outlook clients.

## Task pane and content add-ins

If any content or task pane add-in exceeds the preceding thresholds on CPU core or memory usage, or tolerance limit for crashes, the corresponding Office application displays a warning for the user. At this point, the user can do one of the following:

- Restart the add-in.
- Cancel further alerts about exceeding that threshold. Ideally, the user should then delete the add-in from the document. Continued use of the add-in would risk further performance and stability issues.

## Evaluation response time for regular expressions in Outlook add-ins

Outlook add-ins that use regular expressions and run in Outlook on Windows (classic) or on Mac (classic) should observe the following rules on activation.

- **Regular expressions response time** - A default threshold of 1,000 milliseconds for Outlook to evaluate all regular expressions in the manifest of an Outlook add-in. Exceeding the threshold causes Outlook to retry evaluation at a later time.

In classic Outlook on Windows, administrators can adjust this default threshold value of 1,000 milliseconds by using a group policy or application-specific setting for the

`HKEY_CURRENT_USER\SOFTWARE\Microsoft\Office\16.0\WEF\Outlook\ActivationAlertThreshold` DWORD value in the Windows registry.

- **Regular expressions re-evaluation** - A default limit of three times for Outlook to reevaluate all the regular expressions in a manifest. If evaluation fails three times, the user must switch to a different mail item then switch back to retry evaluation.

Administrators can adjust this number of times to retry evaluation by using a group policy or application-specific setting. The location of the setting depends on the platform.

- **Windows:** The

`HKEY_CURRENT_USER\SOFTWARE\Microsoft\Office\16.0\WEF\Outlook\ActivationRetryLimit` DWORD value in the Windows registry.

- **Mac:** The `ActivationRetryLimit` property list in `~/Library/Preferences`.

## Excel add-ins

Excel add-ins have important data transfer limits when interacting with the workbook.

- Excel on the web has a payload size limit for requests and responses of **5MB**. `RichAPI.Error` will be thrown if that limit is exceeded.
- A range is limited to **5,000,000** cells for read operations.

If you expect user input will exceed these limits, check the data before calling `context.sync()`. Split the operation into smaller pieces as needed. Call `context.sync()` for each sub-operation to avoid those operations getting batched together again.

These limits are typically exceeded by large ranges. Your add-in might be able to use [RangeAreas](#) to strategically update cells within a larger range. For more information about working with `RangeAreas`, see [Work with multiple ranges simultaneously in Excel add-ins](#). For additional information about optimizing payload size in Excel, see [Payload size limit best practices](#).

## Verify resource usage issues in the Telemetry Log


Office provides a Telemetry Log that maintains a record of certain events (loading, opening, closing, and errors) for Office solutions running on the local computer. This includes resource

usage issues in an Office Add-in. If you have the Telemetry Log set up, you can use Excel to open the Telemetry Log in the following default location on your local drive.

```
%Users%\<Current user>\AppData\Local\Microsoft\Office\16.0\Telemetry
```

For each event that the Telemetry Log tracks for an add-in, there is a date/time of the occurrence, event ID, severity, and short descriptive title for the event, the friendly name and unique ID of the add-in, and the application that logged the event. Refresh the Telemetry Log to see the current tracked events.

The following table lists the events that the Telemetry Log tracks for Office Add-ins.

 Expand table

Event ID	Title	Severity	Description
7	Add-in manifest downloaded successfully	<i>Not applicable</i>	The manifest of the Office Add-in was successfully loaded and read by the Office application.
8	Add-in manifest did not download	Critical	The Office application was unable to load the manifest file for the Office Add-in from the SharePoint catalog, corporate catalog, or AppSource.
9	Add-in markup could not be parsed	Critical	The Office application loaded the Office Add-in manifest, but could not read the HTML markup of the app.
10	Add-in used too much CPU	Critical	The Office Add-in used more than 90% of the CPU resources over a finite period of time.
15	Add-in disabled due to string search time-out	<i>Not applicable</i>	Outlook add-ins search the subject line and message of an e-mail to determine whether they should be displayed by using a regular expression. The Outlook add-in listed in the <b>File</b> column was disabled by Outlook because it timed out repeatedly while trying to match a regular expression.
18	Add-in closed successfully	<i>Not applicable</i>	The Office application was able to close the Office Add-in successfully.
19	Add-in encountered runtime error	Critical	The Office Add-in had a problem that caused it to fail. For more details, look at the <b>Microsoft Office Alerts</b> log using the Windows Event Viewer on the computer that encountered the error.
20	Add-in failed to verify licensing	Critical	The licensing information for the Office Add-in could not be verified and may have expired. For more details, look at the

Event ID	Title	Severity	Description
			Microsoft Office Alerts log using the Windows Event Viewer on the computer that encountered the error.

For more information, see [Deploying Telemetry Dashboard](#) and [Troubleshooting Office files and custom solutions with the telemetry log](#).

## Design and implementation techniques

While the resources limits on CPU and memory usage, crash tolerance, and UI responsiveness apply to Office Add-ins running only in Office desktop clients, optimization should be a priority if you want your add-in to perform satisfactorily on all supporting clients and devices.

Optimization is particularly important if your add-in carries out long-running operations or handles large data sets. The following list suggests some techniques to break up CPU-intensive or data-intensive operations into smaller chunks so that your add-in avoids excessive resource consumption and keeps the Office application responsive.

- If your add-in needs to read a large volume of data from an unbounded dataset, you can apply paging when reading the data from a table, or reduce the size of data in each shorter read operation, rather than attempting to complete the read in one single operation. You can do this through the [setTimeout](#) method of the global object to limit the duration of input and output. It also handles the data in defined chunks instead of randomly unbounded data. Another option is to use [async](#) to handle your Promises.
- If your add-in uses a CPU-intensive algorithm to process a large volume of data, you can use [web workers](#) to perform the long-running task in the background while running a separate script in the foreground, such as displaying progress in the user interface. Web workers don't block user activities and allow the HTML page to remain responsive. For an example of web workers, see [The Basics of Web Workers](#).
- If your add-in uses a CPU-intensive algorithm but you can divide the data input or output into smaller sets, consider creating a web service, passing the data to the web service to off-load the CPU, and waiting for an asynchronous callback.
- Test your add-in against the highest volume of data you expect, and restrict your add-in to process up to that limit.

## Performance improvements with the application-specific APIs

The performance tips in [Using the application-specific API model](#) provide guidance when using the application-specific APIs for Excel, OneNote, Visio, and Word. In summary, you should:

- Only load necessary properties.
- Minimize the number of `sync()` calls. Read [Avoid using the context.sync method in loops](#) for further information on how to manage `sync` calls in your code.
- Minimize the number of proxy objects created. You can also untrack proxy objects, as described in the next section.

## Untrack unneeded proxy objects

Proxy objects persist in memory until `RequestContext.sync()` is called. Large batch operations may generate a lot of proxy objects that are only needed once by the add-in and can be released from memory before the batch executes.

The `untrack()` method releases the object from memory. This method is implemented on many application-specific API proxy objects. Calling `untrack()` after your add-in is done with the object should yield a noticeable performance benefit when using large numbers of proxy objects.

### ⓘ Note

`Range.untrack()` is a shortcut for

`ClientRequestContext.trackedObjects.remove(thisRange)`. Any proxy object can be untracked by removing it from the tracked objects list in the context.

The following Excel code sample fills a selected range with data, one cell at a time. After the value is added to the cell, the range representing that cell is untracked. Run this code with a selected range of 10,000 to 20,000 cells, first with the `cell.untrack()` line, and then without it. You should notice the code runs faster with the `cell.untrack()` line than without it. You may also notice a quicker response time afterwards, since the cleanup step takes less time.

JavaScript

```
Excel.run(async (context) => {
    const largeRange = context.workbook.getSelectedRange();
    largeRange.load(["rowCount", "columnCount"]);
    await context.sync();

    for (let i = 0; i < largeRange.rowCount; i++) {
        for (let j = 0; j < largeRange.columnCount; j++) {
            let cell = largeRange.getCell(i, j);
            cell.values = [[i * j]];

            // Call untrack() to release the range from memory.
            cell.untrack();
        }
    }
});
```

```
}  
  
    await context.sync();  
});
```

Note that needing to untrack objects only becomes important when you're dealing with thousands of them. Most add-ins don't need to manage proxy object tracking.

## See also

- [Privacy and security for Office Add-ins](#)
- [Limits for activation and JavaScript API for Outlook add-ins](#)
- [Performance optimization using the Excel JavaScript API](#)