

Support older Microsoft webviews and Office versions

Article • 01/07/2025

Office Add-ins are web applications that are displayed inside iframes when running on Office on the web. Office Add-ins are displayed using an embedded browser control (also known as a webview) when running in Office on Windows or Office on the Mac. The embedded browser controls are supplied by the operating system or by a browser installed on the user's computer.

Important

Webviews from Internet Explorer and Microsoft Edge Legacy are still used in Office Add-ins

Some combinations of platforms and Office versions, including volume-licensed perpetual versions through Office 2019, still use the webview controls that come with Internet Explorer 11 (called "Trident") and Microsoft Edge Legacy (called "EdgeHTML") to host add-ins, as explained in [Browsers and webview controls used by Office Add-ins](#). Internet Explorer 11 was disabled in Windows 10 and Windows 11 in February 2023, and the UI for launching it was removed; but it's still installed on with those operating systems. So, Trident and other functionality from Internet Explorer can still be called programmatically by Office.

We recommend (but don't require) that you support these combinations, at least in a minimal way, by providing users of your add-in a graceful failure message when your add-in is launched in these webviews. Keep these additional points in mind:

- Office on the web no longer opens in Internet Explorer or Microsoft Edge Legacy. Consequently, [AppSource](#) doesn't test add-ins in Office on the web on these browsers.
- AppSource still tests for combinations of platform and Office *desktop* versions that use Trident or EdgeHTML. However, it only issues a warning when the add-in doesn't support these webviews; the add-in isn't rejected by AppSource.
- The [Script Lab tool](#) no longer supports Trident.

If you plan to support older versions of Windows and Office, your add-in must work in the embeddable browser controls used by these versions. For example, browser controls

based on Internet Explorer 11 (IE11) or Microsoft Edge Legacy (EdgeHTML-based). For information about which combinations of Windows and Office use these browser controls, see [Browsers and webview controls used by Office Add-ins](#).

Determine the webview the add-in is running in at runtime

Your add-in can discover the webview that it's running in by reading the [window.navigator.userAgent](#) property. This enables the add-in to either provide an alternate experience or gracefully fail. The following is an example that determines whether the add-in is running in Trident or EdgeHTML.

JavaScript

```
if (navigator.userAgent.indexOf("Trident") !== -1) {
    /*
        Trident is the webview in use. Do one of the following:
        1. Provide an alternate add-in experience that doesn't use any of
the HTML5
        features that aren't supported in Trident (Internet Explorer 11).
        2. Enable the add-in to gracefully fail by adding a message to the
UI that
        says something similar to:
        "This add-in won't run in your version of Office. Please upgrade
either to
        perpetual Office 2021 (or later) or to a Microsoft 365 account."
    */
} else if (navigator.userAgent.indexOf("Edge") !== -1) {
    /*
        EdgeHTML is the browser in use. Do one of the following:
        1. Provide an alternate add-in experience that's supported in
EdgeHTML (Microsoft Edge Legacy).
        2. Enable the add-in to gracefully fail by adding a message to the
UI that
        says something similar to:
        "This add-in won't run in your version of Office. Please upgrade
either to
        perpetual Office 2021 (or later) or to a Microsoft 365 account."
    */
} else {
    /*
        A webview other than Trident or EdgeHTML is in use.
        Provide a full-featured version of the add-in here.
    */
}
```

📌 Note

Microsoft Edge (Chromium) returns `edg/` followed by one or more version digits and zero or more `.` separators as the user agent; for example, `edg/76.0.167.1`. **Note that the `e` isn't present at the end of name! It's "edg", not "edge".**

This JavaScript should be as early in the add-in startup process as possible. The following is an example of an add-in home page that advises users to upgrade Office when Trident is detected.

HTML

```
<!doctype html>
<html lang="en" data-framework="typescript">

<head>
  <meta charset="UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=Edge" />
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Contoso Task Pane Add-in</title>

  <!-- Office JavaScript API -->
  <script type="text/javascript"
src="https://appsforoffice.microsoft.com/lib/1/hosted/office.js"></script>
</head>

<body>
  <div id="main">
    <!--
      The add-in UI is here.
    -->
  </div>


  <!--
    The script below makes the following div display if the
    webview is Trident, and hides the regular div.
  -->
  <div id="tridentmessage" style="display: none; padding: 10px;">
    This add-in will not run in your version of Office. Please upgrade
    either to
    perpetual Office 2021 (or later) or to a Microsoft 365 account.
  </div>
  <script>
    if (navigator.userAgent.indexOf("Trident") !== -1) {
      var tridentMessage = document.getElementById("tridentmessage");
      var normalUI = document.getElementById("main");
      tridentMessage.style.display = "block";
      normalUI.style.display = "none";
    }
  </script>
</body>
</html>
```

Important

It's not always a good practice to read the `userAgent` property. Be sure you're familiar with the article, [Browser detection using the user agent](#)[↗], including the recommendations and alternatives to reading `userAgent`. In particular, if you're providing an alternate add-in experience to support the use of Trident, consider using feature detection instead of testing for the user agent. But if you're just providing a notification that the add-in doesn't work in Trident, as in this case, using `userAgent` is appropriate.


As of July 24th, 2023, the non-English versions of the article are all out-of-date to varying degrees; some are over 12 years out-of-date.

As of the same date, the text and tables in the section [Which part of the user agent contains the information you are looking for?](#)[↗] of the *English* version of the article no longer mention Trident or Internet Explorer 11. In the table for **Browser Name and version**, the row for Internet Explorer 11 was the following:

 Expand table

Engine	Must contain	Must not contain
Internet Explorer 11	<code>Trident/7.0; .*rv:xyz</code>	

In the table for **Rendering engine**, the row for Trident was the following:

 Expand table

Engine	Must contain	Comment
Trident	<code>Trident/xyz</code>	Internet Explorer places this fragment in the comments section of the User-Agent string.

Review webview and Office version support information

For more information on how to support specific webviews and Office versions, select the applicable tab.

Trident (Internet Explorer)

The JavaScript engine associated with Trident doesn't support JavaScript versions later than ES5. To use more modern versions of JavaScript or to use TypeScript, see [Support for recent versions of JavaScript](#).

Important

Trident doesn't support some HTML5 features such as media, recording, and location. If your add-in must support Trident, then you must either design the add-in to avoid these unsupported features or the add-in must detect when Trident is being used and provide an alternate experience that doesn't use the unsupported features. For more information, see [Determine the webview the add-in is running in at runtime](#).

Test an add-in on Trident (Internet Explorer)

See [Trident testing](#).

Support for recent versions of JavaScript

If you want to use the syntax and features of a version of JavaScript that is newer than the one supported in the webview or runtime that your code is running in, or you want to use TypeScript, you must use a transpiler or a polyfill or both. For example, a transpiler will convert syntax or operators, such as the `=>` operator, that is unknown in ES5, to ES5. A polyfill replaces methods, types, and classes from a newer version of JavaScript into equivalent functionality in an older version.

The following subsections assume that the target JavaScript standard is ES5, but the information applies with other targets too. For example, if your target is ECMAScript 2016, just replace "ES5" with "ECMAScript 2016" (and "post-ES5" with "post-ECMAScript 2016") in these subsections.

Use a transpiler

You can write your code in either TypeScript or modern JavaScript and then transpile it at build-time into ES5 JavaScript. The resulting ES5 files are what you upload to your add-in's web application.

There are two popular transpilers. Both of them can work with source files that are TypeScript or post-ES5 JavaScript. They also work with React files (.jsx and .tsx).

- [babel](#)
- [tsc](#)

See the documentation for either of them for information about installing and configuring the transpiler in your add-in project. We recommend that you use a task runner, such as [Grunt](#) or [Webpack](#) to automate the transpilation. For a sample add-in that uses tsc, see [Office Add-in Microsoft Graph React](#). For a sample that uses babel, see [Offline Storage Add-in](#).

ⓘ Note

If you're using Visual Studio (not Visual Studio Code), tsc is probably easiest to use. You can install support for it with a nuget package. For more information, see [JavaScript and TypeScript in Visual Studio](#). To use babel with Visual Studio, create a build script or use the Task Runner Explorer in Visual Studio with tools like the [Webpack Task Runner](#) or [NPM Task Runner](#).

Use a polyfill

A [polyfill](#) is earlier-version JavaScript that duplicates functionality from more recent versions of JavaScript. The polyfill works in webviews that don't support the later JavaScript versions. For example, the string method `startsWith` wasn't part of the ES5 version of JavaScript, and so it won't run in Trident (Internet Explorer 11). There are polyfill libraries, written in ES5, that define and implement a `startsWith` method. We recommend the [core-js](#) polyfill library.

To use a polyfill library, load it like any other JavaScript file or module. For example, you can use a `<script>` tag in the add-in's home page HTML file (for example `<script src="/js/core-js.js"></script>`), or you can use an `import` statement in a JavaScript file (for example, `import 'core-js';`). When the JavaScript engine sees a method like `startsWith`, it will first look to see if there's a method of that name built into the language. If there is, it will call the native method. If, and only if, the method isn't built-in, the engine will look in all loaded files for it. So, the polyfilled version isn't used in browsers that support the native version.

Importing the entire core-js library will import all core-js features. You can also import only the polyfills that your Office Add-in requires. For instructions about how to do this, see [CommonJS APIs](#). The core-js library has most of the polyfills that you need. There are a few exceptions detailed in the [Missing Polyfills](#) section of the core-js

documentation. For example, it doesn't support `fetch`, but you can use the [fetch](#) polyfill.

For a sample add-in that uses core.js, see [Word Add-in Angular2 StyleChecker](#).

See also

- [Browsers and webview controls used by Office Add-ins](#)
- [ECMAScript 6 compatibility table](#)
- [Can I use... Support tables for HTML5, CSS3, etc](#)