Localization for Office Add-ins

Article • 02/12/2025

You can implement any localization scheme that's appropriate for your Office Add-in. The JavaScript API and manifest schema of the Office Add-ins platform provide some choices. You can use the Office JavaScript API to determine a locale and display strings based on the locale of the Office application, or to interpret or display data based on the locale of the data. You can use the manifest to specify locale-specific add-in file location and descriptive information. Alternatively, you can use Visual Studio and Microsoft Ajax script to support globalization and localization.

Use the JavaScript API to determine locale-specific strings

The Office JavaScript API provides two properties that support displaying or interpreting values consistent with the locale of the Office application and data.

- Context.displayLanguage specifies the locale (or language) of the user interface of the Office application. The following example verifies if the Office application uses the en-US or fr-FR locale, and displays a locale-specific greeting.

  JavaScript

  ```javascript
  function sayHelloWithDisplayLanguage() {
      const myLanguage = Office.context.displayLanguage;
      switch (myLanguage) {
          case 'en-US':
              write('Hello!');
              break;
          case 'fr-FR':
              write('Bonjour!');
              break;
      }
  }

  // Function that writes to a div with id='message' on the page.
  function write(message) {
      document.getElementById('message').innerText += message;
  }
  ```

- Context.contentLanguage specifies the locale (or language) of the data. Extending the last code sample, instead of checking the displayLanguage property, assign `myLanguage` the value of the contentLanguage property, and use the rest of the same code to display a greeting based on the locale of the data.

JavaScript

```javascript
const myLanguage = Office.context.contentLanguage;
```

Control localization from the manifest

The techniques for localizing with the manifest differ depending on whether you're using the add-in only manifest or the unified app manifest for Microsoft 365.

Unified app manifest for Microsoft 365

When using the unified app manifest for Microsoft 365, localize the public-facing strings in the manifest as described in Localize strings in your app manifest. The following is an example for an Outlook add-in. First is the "localizationInfo" object in the manifest. Below that is the fr-fr.json file with the translated strings. The add-in has a task pane (with a French version of the home page), localized French icons, and a custom ribbon button that opens a video player in a dialog box.

JSON

```json
"localizationInfo": {
  "defaultLanguageTag": "en",
  "additionalLanguages": [
    {
      "languageTag": "fr-fr",
      "file": "fr-fr.json"
    }
  ]
}
```

JSON

```json
{
  "$schema": "https://developer.microsoft.com/json-
schemas/teams/v1.16/MicrosoftTeams.Localization.schema.json",
  "name.short": "Lecteur vidéo",
  "name.full": "Lecteur vidéo pour Outlook",
  "description.short": "Voir les vidéos YouTube dans Outlook via les
mails.",
  "description.full": "Visualisez les vidéos YouTube référencées dans
vos courriers électronique directement depuis Outlook.",
  "icons.color": "https://localhost:3000/assets/fr-fr/icon-128.png",
  "extensions[0].audienceClaimUrl": "https://localhost:3000/fr-
fr/taskpane.html",
    "extensions[0].ribbons[0].tabs[0].groups[0].label": "Outils de
médias",
    "extensions[0].ribbons[0].tabs[0].groups[0].controls[0].icons[0].url":
"https://localhost:3000/assets/fr-fr/player-icon.png",
    "extensions[0].ribbons[0].tabs[0].groups[0].controls[0].label":
"Ouvrir le lecteur vidéo",

"extensions[0].ribbons[0].tabs[0].groups[0].controls[0].supertip.descrip
tion": "Cliquez pour ouvrir le lecteur vidéo.",

"extensions[0].ribbons[0].tabs[0].groups[0].controls[0].supertip.title":
"Ouvrir le lecteur vidéo",
}
```

Match date/time format with client locale

You can get the locale of the user interface of the Office client application by using the **displayLanguage** property. You can then display date and time values in a format consistent with the current locale of the Office application. One way to do that is to prepare a resource file that specifies the date/time display format to use for each locale that your Office Add-in supports. At run time, your add-in can use the resource file and match the appropriate date/time format with the locale obtained from the **displayLanguage** property.

You can get the locale of the data of the Office client application by using the contentLanguage property. Based on this value, you can then appropriately interpret or display date/time strings. For example, the `jp-JP` locale expresses data/time values as `yyyy/MM/dd`, and the `fr-FR` locale, `dd/MM/yyyy`.

Use Visual Studio to create a localized and globalized add-in

If you use Visual Studio to create Office Add-ins, the .NET Framework and Ajax provide ways to globalize and localize client script files.

You can globalize and use the Date and Number JavaScript type extensions and the JavaScript Date object in the JavaScript code for an Office Add-in to display values based on the locale settings on the current browser. For more information, see Walkthrough: Globalizing a Date by Using Client Script.

You can include localized resource strings directly in standalone JavaScript files to provide client script files for different locales, which are set on the browser or provided by the user. Create a separate script file for each supported locale. In each script file, include an object in JSON format that contains the resource strings for that locale. The localized values are applied when the script runs in the browser.

Example: Build a localized Office Add-in

This section provides examples that show you how to localize an Office Add-in description, display name, and UI.

> **Note**
>
> To download Visual Studio, see the Visual Studio IDE page. During installation you'll need to select the Office/SharePoint development workload.

Configure Office to use additional languages for display or editing

To run the sample code provided, configure Office on your computer to use additional languages so that you can test your add-in by switching the language used for display in menus and commands, for editing and proofing, or both.

You can use an Office Language pack to install an additional language. For more information about Language Packs and where to get them, see Language Accessory Pack for Office.

After you install the Language Accessory Pack, you can configure Office to use the installed language for display in the UI, for editing document content, or both. The example in this article uses an installation of Office that has the Spanish Language Pack applied.

Create an Office Add-in project

You'll need to create a Visual Studio Office Add-in project.

> **Note**
>
> If you haven't installed Visual Studio, see the Visual Studio IDE page for download instructions. During installation you'll need to select the Office/SharePoint development workload. If you've previously installed Visual Studio 2019 or later, use the Visual Studio Installer to ensure that the Office/SharePoint development workload is installed.

1. Choose **Create a new project**.

2. Using the search box, enter **add-in**. Choose **Word Web Add-in**, then select **Next**.

3. Name your project **WorldReadyAddIn** and select **Create**.

4. Visual Studio creates a solution and its two projects appear in **Solution Explorer**. The **Home.html** file opens in Visual Studio.

Localize the text used in your add-in

The text that you want to localize for another language appears in two areas.

- **Add-in display name and description**. This is controlled by entries in the add-in manifest file.

- **Add-in UI**. You can localize the strings that appear in your add-in UI by using JavaScript code, for example, by using a separate resource file that contains the localized strings.

Localize the add-in display name and description

1. In **Solution Explorer**, expand **WorldReadyAddIn**, **WorldReadyAddInManifest**, and then choose **WorldReadyAddIn.xml**.

2. In **WorldReadyAddInManifest.xml**, replace the DisplayName and Description elements with the following block of code.

   ⓘ **Note**

   You can replace the Spanish language localized strings used in this example for the DisplayName and Description elements with the localized strings for any other language.

   XML

```xml
<DisplayName DefaultValue="World Ready add-in">
  <Override Locale="es-es" Value="Aplicación de uso internacional"/>
</DisplayName>
<Description DefaultValue="An add-in for testing localization">
  <Override Locale="es-es" Value="Una aplicación para la prueba de la
localización"/>
</Description>
```

3. When you change the display language for Microsoft 365 from English to Spanish, for example, and then run the add-in, the add-in display name and description are shown with localized text.

Lay out the add-in UI

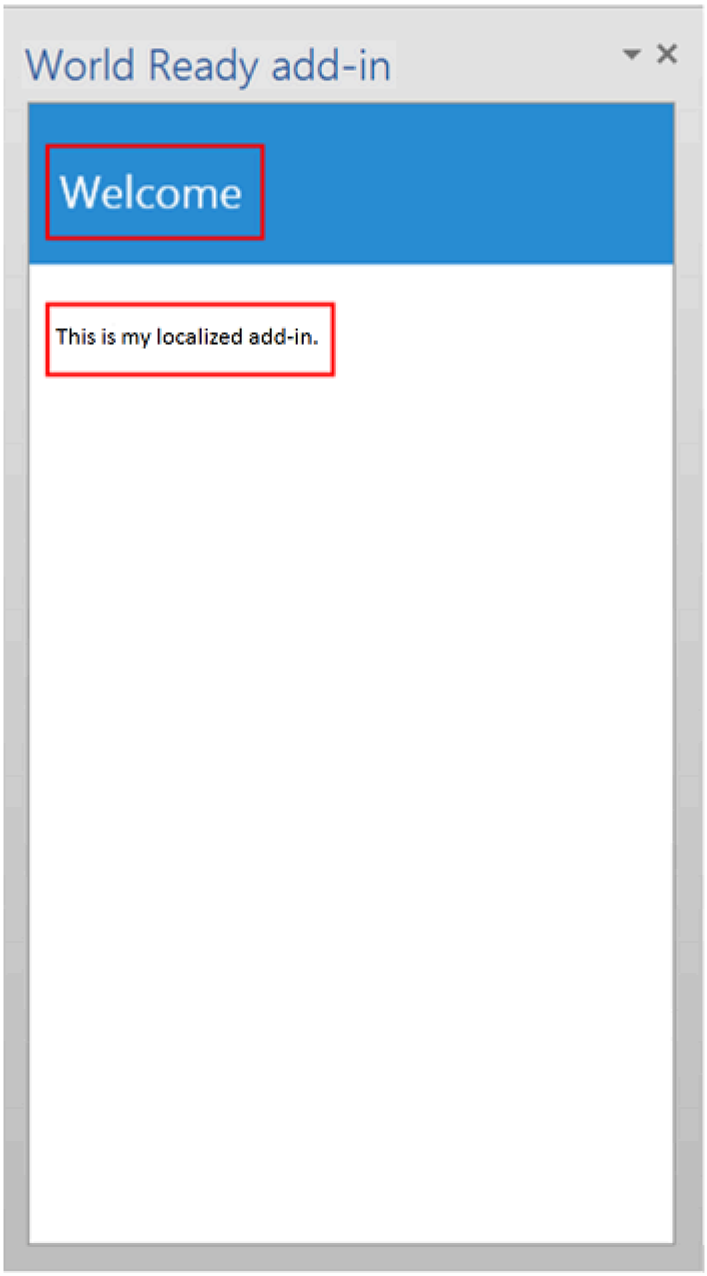
1. In Visual Studio, in **Solution Explorer**, choose **Home.html**.

2. Replace the `<body>` element contents in **Home.html** with the following HTML, and save the file.

HTML

```html
<body>
    <!-- Page content -->
    <div id="content-header" class="ms-bgColor-themePrimary ms-font-
xl">
        <div class="padding">
            <h1 id="greeting" class="ms-fontColor-white"></h1>
        </div>
    </div>
    <div id="content-main">
        <div class="padding">
            <div class="ms-font-m">
                <p id="about"></p>
            </div>
        </div>
    </div>
</body>
```

The following figure shows the heading (h1) element and the paragraph (p) element that will display localized text when you complete the remaining steps and run the add-in.



Add the resource file that contains the localized strings

The JavaScript resource file contains the strings used for the add-in UI. The HTML for the sample add-in UI contains an `<h1>` element that displays a greeting, and a `<p>` element that introduces the add-in to the user.

To enable localized strings for the heading and paragraph, you place the strings in a separate resource file. The resource file creates a JavaScript object that contains a separate JavaScript Object Notation (JSON) object for each set of localized strings. The resource file also provides a method for getting back the appropriate JSON object for a given locale.

Add the resource file to the add-in project

1. In **Solution Explorer** in Visual Studio, right-click (or select and hold) the **WorldReadyAddInWeb** project, then choose **Add** > **New Item**.

2. In the **Add New Item** dialog box, choose **JavaScript File**.

3. Enter **UIStrings.js** as the file name and choose **Add**.

4. Add the following code to the **UIStrings.js** file, and save the file.

JavaScript

```javascript
/* Store the locale-specific strings */

const UIStrings = (() => {
    "use strict";

    const UIStrings = {};

    // JSON object for English strings
    UIStrings.EN = {
        "Greeting": "Welcome",
        "Introduction": "This is my localized add-in."
    };

    // JSON object for Spanish strings
    UIStrings.ES = {
        "Greeting": "Bienvenido",
        "Introduction": "Esta es mi aplicación localizada."
    };

    UIStrings.getLocaleStrings = (locale) => {
        let text;

        // Get the resource strings that match the language.
        switch (locale) {
            case 'en-US':
                text = UIStrings.EN;
                break;
            case 'es-ES':
                text = UIStrings.ES;
                break;
            default:
                text = UIStrings.EN;
                break;
        }

        return text;
    };

    return UIStrings;
})();
```

The **UIStrings.js** resource file creates an object, **UIStrings**, which contains the localized strings for your add-in UI.

Localize the text used for the add-in UI

To use the resource file in your add-in, you'll need to add a script tag for it on **Home.html**. When **Home.html** is loaded, **UIStrings.js** executes and the **UIStrings** object that you use to get the strings is available to your code. Add the following HTML in the head tag for **Home.html** to make **UIStrings** available to your code.

HTML

```html
<!-- Resource file for localized strings: -->
<script src="../UIStrings.js" type="text/javascript"></script>
```

Now you can use the **UIStrings** object to set the strings for the UI of your add-in.

If you want to change the localization for your add-in based on what language is used for display in menus and commands in the Office client application, you use the **Office.context.displayLanguage** property to get the locale for that language. For example, if the application language uses Spanish for display in menus and commands, the **Office.context.displayLanguage** property will return the language code es-ES.

If you want to change the localization for your add-in based on what language is being used for editing document content, you use the **Office.context.contentLanguage** property to get the locale for that language. For example, if the application language uses Spanish for editing document content, the **Office.context.contentLanguage** property will return the language code es-ES.

After you know the language the application is using, you can use **UIStrings** to get the set of localized strings that matches the application language.

Replace the code in the **Home.js** file with the following code. The code shows how you can change the strings used in the UI elements on **Home.html** based on either the display language of the application or the editing language of the application.

ⓘ Note

To switch between changing the localization of the add-in based on the language used for editing, uncomment the line of code `const myLanguage = Office.context.contentLanguage;` and comment out the line of code `const myLanguage = Office.context.displayLanguage;`

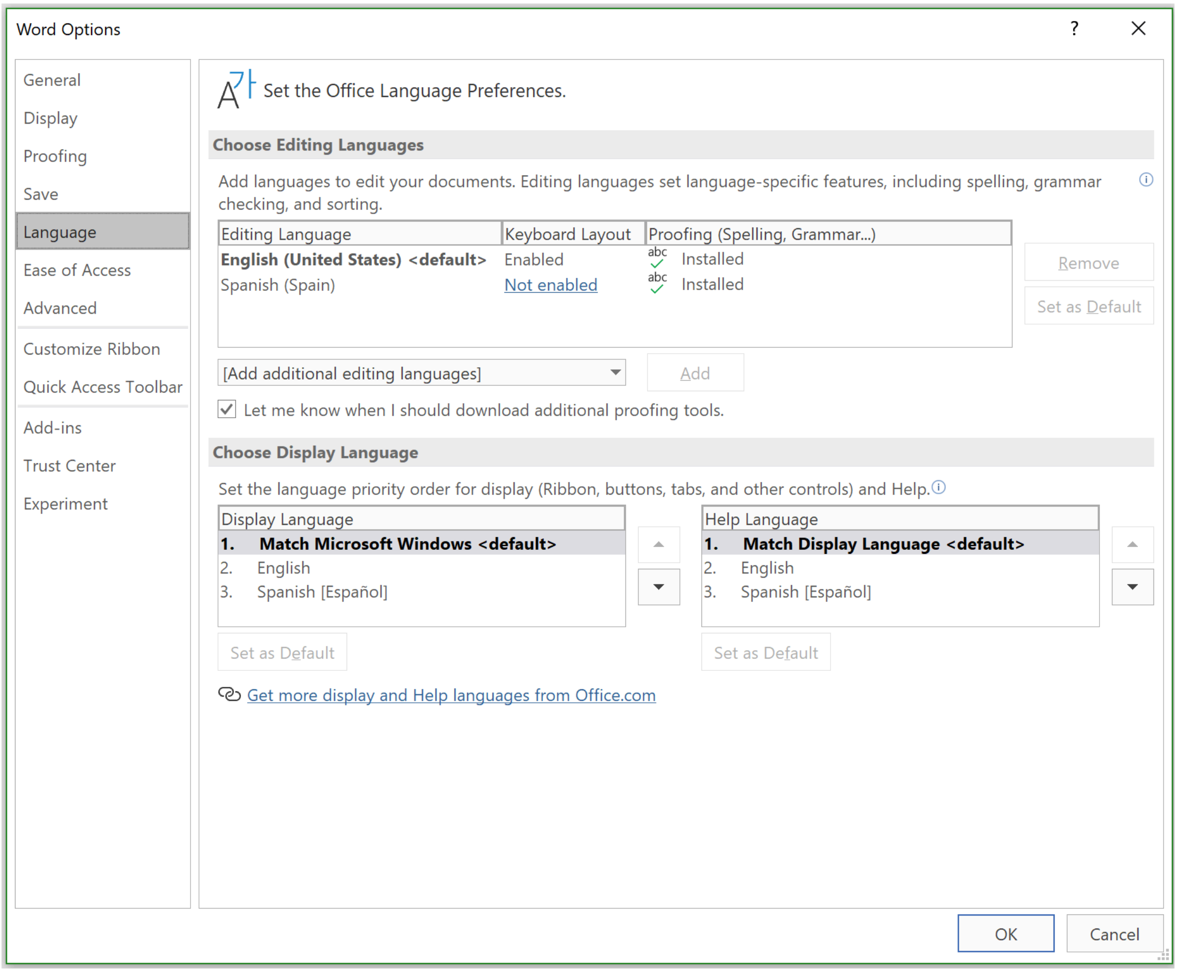
JavaScript

```javascript
/// <reference path="../App.js" />
/// <reference path="../UIStrings.js" />


(() => {
    "use strict";

    // The initialize function must be run each time a new page is loaded.
    Office.onReady(() => {
        $(document).ready(() => {
            // Get the language setting for editing document content.
            // To test this, uncomment the following line and then comment 
out the
            // line that uses Office.context.displayLanguage.
            // const myLanguage = Office.context.contentLanguage;

            // Get the language setting for UI display in the Office 
application.
            const myLanguage = Office.context.displayLanguage;
            let UIText;

            // Get the resource strings that match the language.
            // Use the UIStrings object from the UIStrings.js file
            // to get the JSON object with the correct localized strings.
            UIText = UIStrings.getLocaleStrings(myLanguage);

            // Set localized text for UI elements.
            $("#greeting").text(UIText.Greeting);
            $("#about").text(UIText.Introduction);
        });
    });
})();
```

Test your localized add-in

To test your localized add-in, change the language used for display or editing in the Office application and then run your add-in.

1. In Word, choose **File** > **Options** > **Language**. The following figure shows the **Word Options** dialog box opened to the Language tab.


Word Options

Set the Office Language Preferences.

Choose Editing Languages

Add languages to edit your documents. Editing languages set language-specific features, including spelling, grammar checking, and sorting.

Editing Language	Keyboard Layout	Proofing (Spelling, Grammar...)
English (United States) <default>	Enabled	Installed
Spanish (Spain)	Not enabled	Installed

[Add additional editing languages]

☑ Let me know when I should download additional proofing tools.

Choose Display Language

Set the language priority order for display (Ribbon, buttons, tabs, and other controls) and Help.

Display Language	Help Language
1. Match Microsoft Windows <default>	**1. Match Display Language <default>**
2. English	2. English
3. Spanish [Español]	3. Spanish [Español]

Get more display and Help languages from Office.com

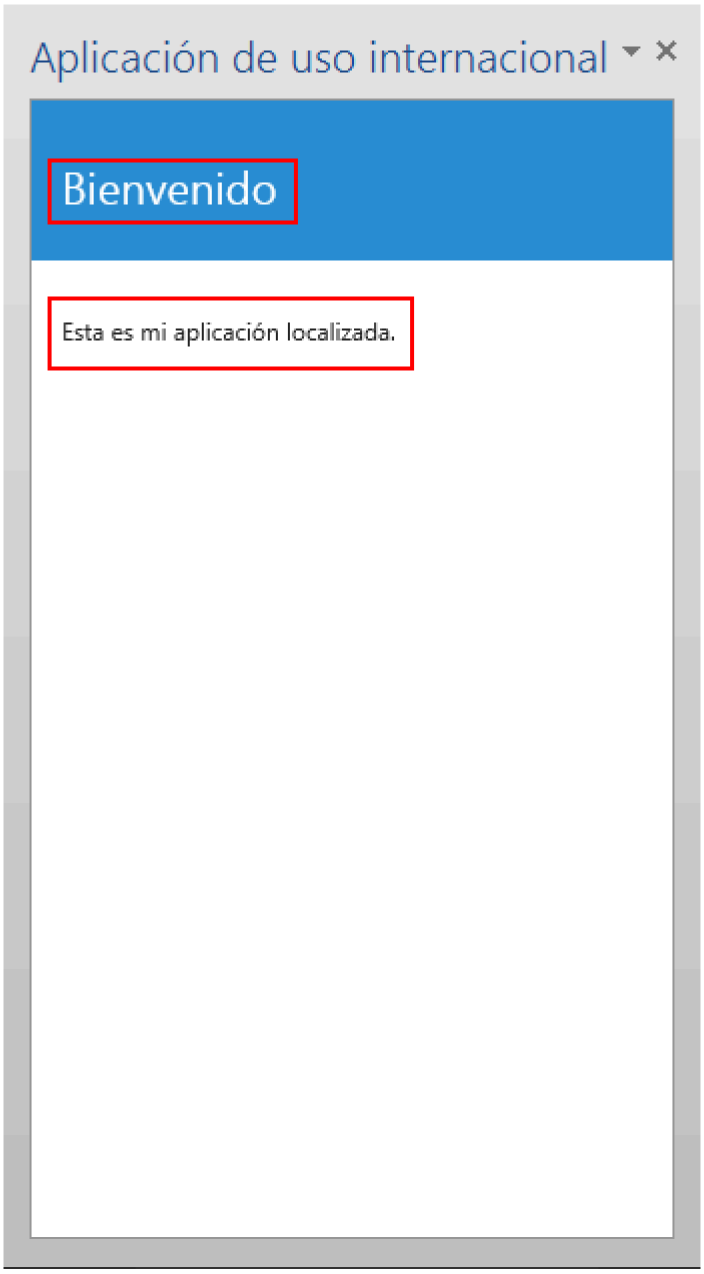
2. Under **Choose Display Language**, select the language that you want for display, for example Spanish, and then choose the up arrow to move the Spanish language to the first position in the list. Alternatively, to change the language used for editing, under **Choose Editing Languages**, choose the language you want to use for editing, for example, Spanish, and then choose **Set as Default**.

3. Choose **OK** to confirm your selection, and then close Word.

4. Press `F5` in Visual Studio to run the sample add-in, or choose **Debug** > **Start Debugging** from the menu bar.

5. In Word, choose **Home** > **Show Taskpane**.

Once running, the strings in the add-in UI change to match the language used by the application, as shown in the following figure.


Aplicación de uso internacional
Bienvenido
Esta es mi aplicación localizada.

See also

- Design guidelines for Office Add-ins
- Overview of deploying languages for Microsoft 365 Apps