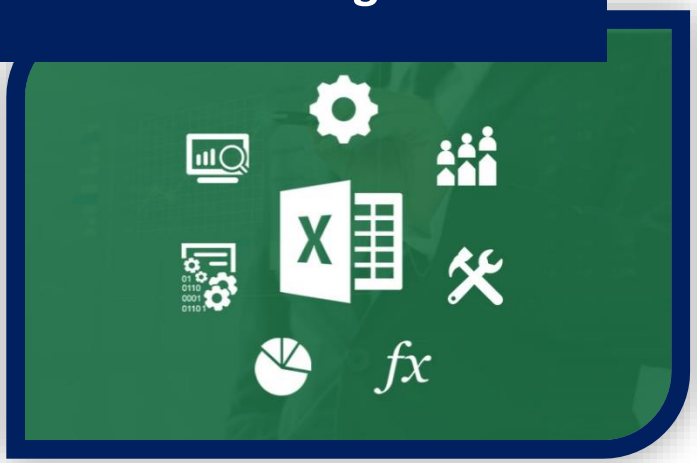
Microsoft Office 365 Excel - Web Add-in - Consulting Practice



Cognitive Convergence is Subject Matter Expert in Office 365, Dynamics 365, SharePoint, Project Server, Power Platform: Power Apps-Power BI-Power Automate-Power Virtual Agents.

Our Microsoft Office 365 Consulting, add-in Development, Customization, Integration services and solutions, can help companies maximize business performance, overcoming market challenges, achieving profitability and providing best customer service.

CONTENTS

OBJECTIVE

In this case study, a brief introduction will be discussed about the Microsoft Excel Web Add-in Practices, its basic structure & architecture, components & the benefits of using the Excel web add-in

INTRODUCTION

An Excel add-in allows the users to extend Excel application functionality across multiple platforms including Windows, Mac, iPad, and in a browser. Use Excel add-ins within a workbook to:

- Interact with Excel objects, read and write Excel data.
- Extend functionality using web-based task pane or content pane
- Add custom ribbon buttons or contextual menu items
- Add custom functions
- Provide richer interaction using a dialog window

BENEFITS OF USING EXCEL ADD-IN

The Office Add-ins platform provides the framework and Office.js JavaScript APIs that enable the users to create and run Excel add-ins. By using the Office Add-ins platform to create the required Excel add-in, users will get the following benefits:

- **Cross-platform support**: Excel add-ins run in Office on the web, Windows, Mac, and iPad.
- **Centralized deployment**: Admins can quickly and easily deploy Excel add-ins to users throughout an organization.
- **Use of standard web technology**: Create your Excel add-in using familiar web technologies such as HTML, CSS, and JavaScript.
- **Distribution via AppSource**: Share your Excel add-in with a broad audience by publishing it to AppSource.

COMPONENTS OF AN EXCEL ADD-IN

An Excel add-in includes two basic components: a web application and a configuration file called a manifest file.

The web application uses the Office JavaScript API to interact with objects in Excel, and can also facilitate interaction with online resources. For example, an add-in can perform any of the following tasks:

- Create, read, update, and delete data in the workbook (worksheets, ranges, tables, charts, named items, and more).
- Perform user authorization with an online service by using the standard OAuth 2.0 flow.
- Issue API requests to Microsoft Graph or any other API.

The web application can be hosted on any web server and can be built using client-side frameworks (such as Angular, React, jQuery) or server-side technologies (such as ASP.NET, Node.js, PHP).

The manifest is an XML configuration file that defines how the add-in integrates with Office clients by specifying settings and capabilities such as:

- The URL of the add-in's web application.
- The add-in's display name, description, ID, version, and default locale.
- How the add-in integrates with Excel, including any custom UI that the add-in creates (ribbon buttons, context menus, and so on).
- Permissions that the add-in requires, such as reading and writing to the document.

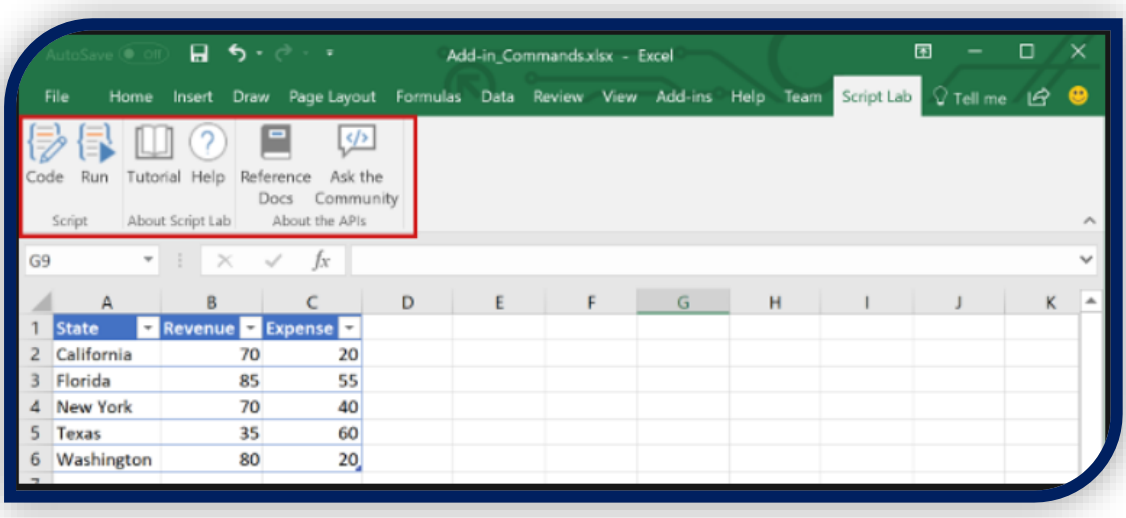
To enable end-users to install and use an Excel add-in, you must publish its manifest either to AppSource or to an add-ins catalog.

CAPABILITIES OF AN EXCEL ADD-IN

In addition to interacting with the content in the workbook, Excel add-ins can add custom ribbon buttons or menu commands, insert task panes, add custom functions, open dialog boxes, and even embed rich, web-based objects such as charts or interactive visualizations within a worksheet.

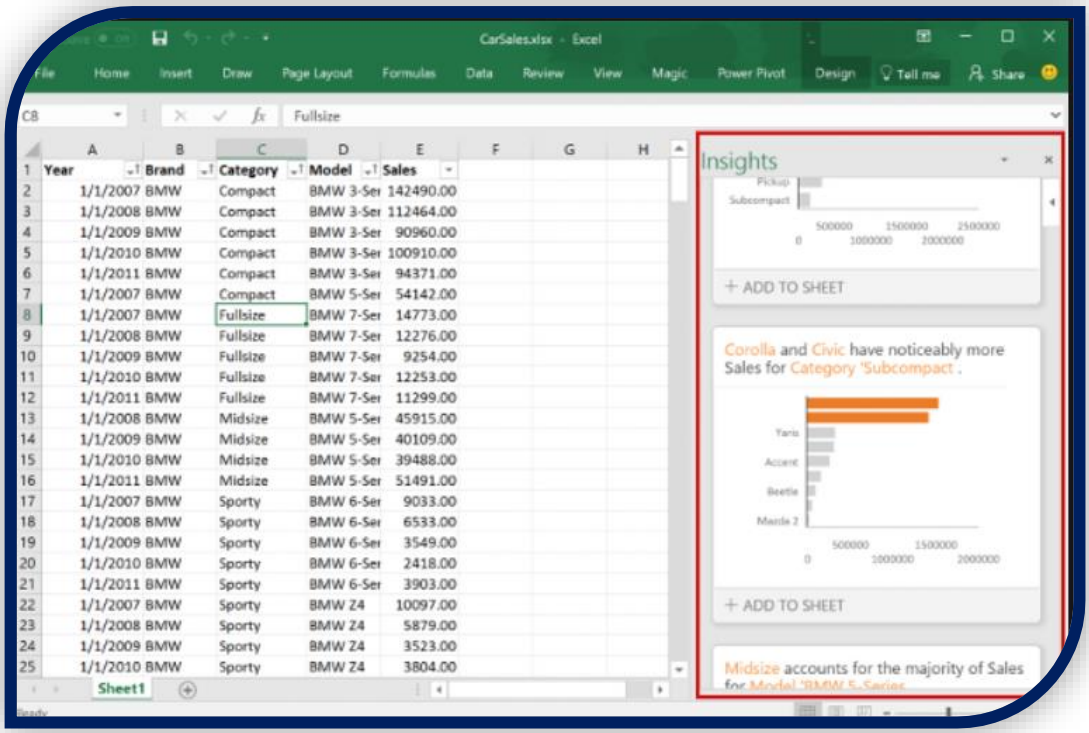
Add-in commands

Add-in commands are UI elements that extend the Excel UI and start actions in your add-in. You can use add-in commands to add a button on the ribbon or an item to a context menu in Excel. When users select an add-in command, they initiate actions such as running JavaScript code or showing a page of the add-in in a task pane.



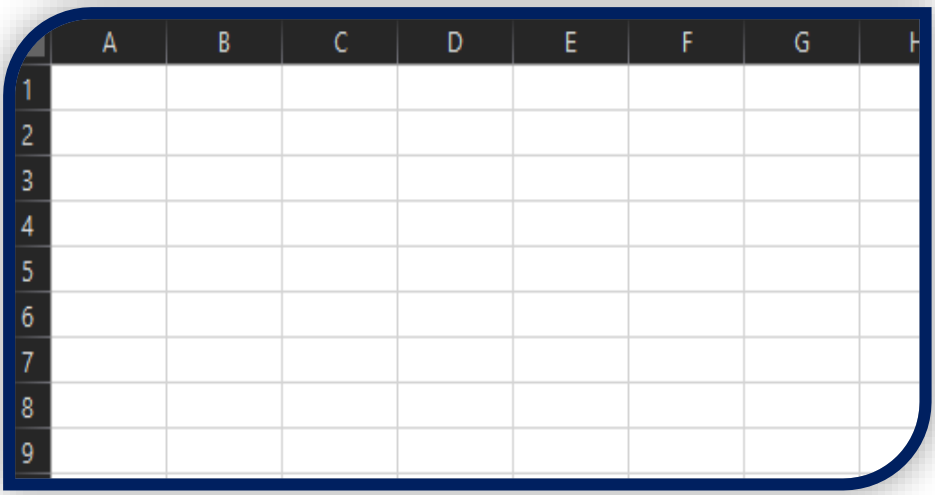
Task panes

Task panes are interface surfaces that typically appear on the right side of the window within Excel. Task panes give users access to interface controls that run code to modify the Excel document or display data from a data source.



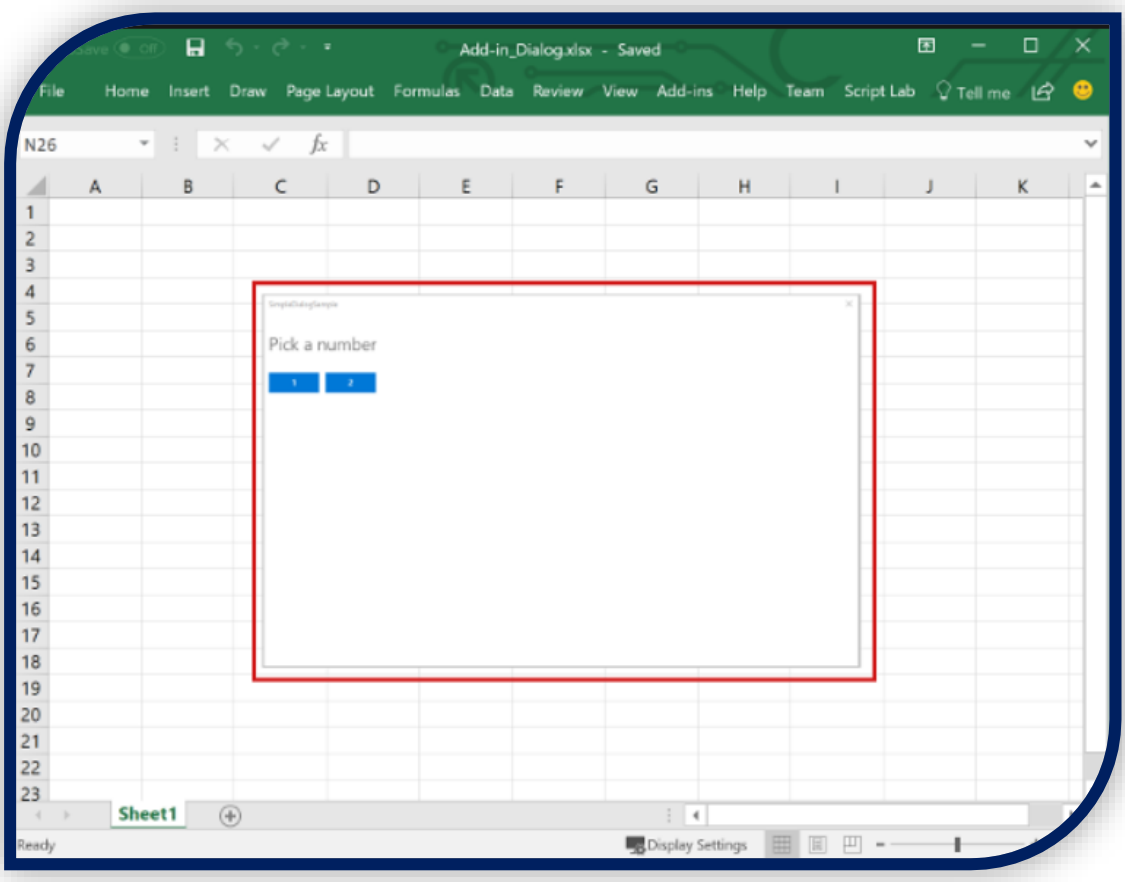
Custom functions

Custom functions enable developers to add new functions to Excel by defining those functions in JavaScript as part of an add-in. Users within Excel can access custom functions just as they would any native function in Excel, such as SUM().



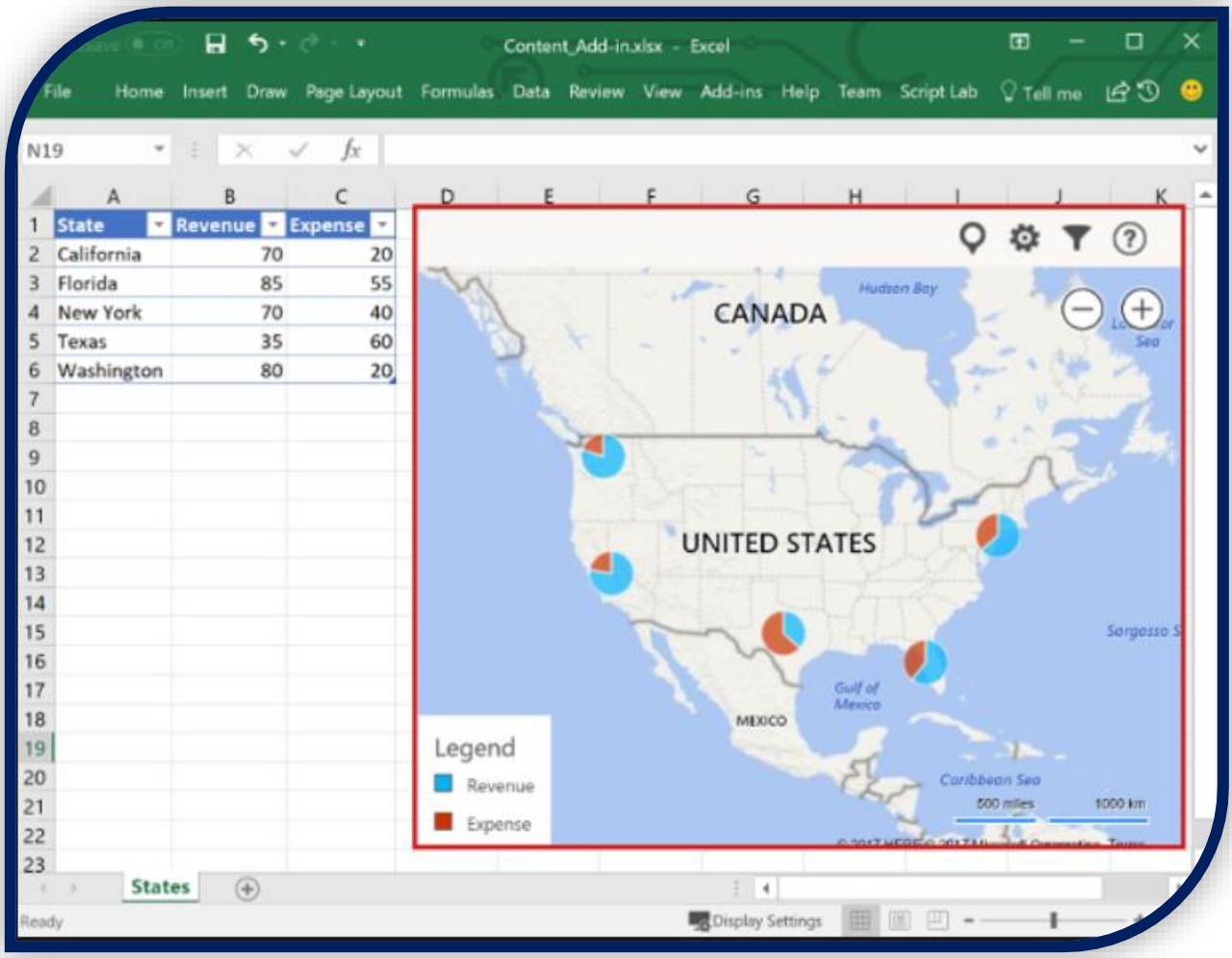
Dialog boxes

Dialog boxes are surfaces that float above the active Excel application window. You can use dialog boxes for tasks such as displaying sign-in pages that can't be opened directly in a task pane, requesting that the user confirm an action, or hosting videos that might be too small if confined to a task pane.



Content add-ins

Content add-ins are surfaces that you can embed directly into Excel documents. You can use content add-ins to embed rich, web-based objects such as charts, data visualizations, or media into a worksheet or to give users access to interface controls that run code to modify the Excel document or display data from a data source. Use content add-ins when you want to embed functionality directly into the document.



EXCEL JAVASCRIPT API

This part describes the core concepts that are fundamental to using the API and guides performing specific tasks such as reading or writing to a large range, updating all cells in the range, and more.

Asynchronous nature of Excel APIs

The web-based Excel add-ins run inside a browser container that is embedded within the Office application on desktop-based platforms such as Office on Windows and runs inside an HTML iFrame in Office on the web. Enabling the Office.js API to interact synchronously with the Excel host across all supported platforms is not feasible due to performance considerations. Therefore, the sync() API call in Office.js returns a promise that is resolved when the Excel application completes the requested read or write actions.

Excel.run

Excel.run executes a function where you specify the actions to perform against the Excel object model. Excel.run automatically creates a request context that you can use to interact with Excel objects. When Excel.run completes, a promise is resolved, and any objects that were allocated at runtime are automatically released. The following example shows how to use Excel.run. The catch statement catches and logs errors that occur within the Excel.run.

```javascript
Excel.run(function (context) {
    // You can use the Excel JavaScript API here in the batch function
    // to execute actions on the Excel object model.
    console.log('Your code goes here.');
}).catch(function (error) {
    console.log('error: ' + error);
    if (error instanceof OfficeExtension.Error) {
        console.log('Debug info: ' + JSON.stringify(error.debugInfo));
    }
});
```

Run options

Excel.run has an overload that takes in a RunOptions object. This contains a set of properties that affect platform behavior when the function runs. The following property is currently supported:

- delayForCellEdit: Determines whether Excel delays the batch request until the user exits cell edit mode. When true, the batch request is delayed and runs when the user exits cell edit mode. When false, the batch request automatically fails if the user is in cell edit mode (causing an error to reach the user). The default behavior with no delayForCellEdit property specified is equivalent to when it is **false**.

```
JavaScript

Excel.run({ delayForCellEdit: true }, function (context) { ... })
```

Request context

Excel and the developed add-in run in two different processes. Since they both use different runtime environments, Excel add-ins require a RequestContext object to connect the add-in to objects in Excel such as worksheets, ranges, charts, and tables.

PROXY OBJECTS

The Excel JavaScript objects that users declare and use in an add-in are proxy objects. Any methods that user invoke or properties that user set or load on proxy objects are simply added to a queue of pending commands. When users call the sync() method on the request context (for example, context.sync()), the queued commands are dispatched to Excel and run. The Excel JavaScript API is fundamentally batch-centric. Users can queue up as many changes as you wish on the request context, and then call the sync() method to run the batch of queued commands.

For example, the following code snippet declares the local JavaScript object selected range to reference a selected range in the Excel document and then sets some properties on that object. A selected range object is a proxy object, so the properties that are set and method that is invoked on that object will not be reflected in the Excel document until your add-in calls context.sync().

JavaScript

```
var selectedRange = context.workbook.getSelectedRange();
selectedRange.format.fill.color = "#4472C4";
selectedRange.format.font.color = "white";
selectedRange.format.autofitColumns();
```

sync()

Calling the sync() method on the request context synchronizes the state between proxy objects and objects in the Excel document. The sync() method runs any commands that are queued on the request context and retrieves values for any properties that should be loaded on the proxy objects. The sync() method executes asynchronously and returns a promise, which is resolved when the sync() method completes.

The following example shows a batch function that defines a local JavaScript proxy object (selectedRange), loads a property of that object, and then uses the JavaScript Promises pattern to call context.sync() to synchronize the state between proxy objects and objects in the Excel document.

JavaScript

```
Excel.run(function (context) {
    var selectedRange = context.workbook.getSelectedRange();
    selectedRange.load('address');
    return context.sync()
      .then(function () {
        console.log('The selected range is: ' + selectedRange.address);
    });
}).catch(function (error) {
    console.log('error: ' + error);
    if (error instanceof OfficeExtension.Error) {
        console.log('Debug info: ' + JSON.stringify(error.debugInfo));
    }
});
```

Load()

Before the reading properties of a proxy object, the user must explicitly load the properties to populate the proxy object with data from the Excel document, and the For example, if the user creates a proxy object to reference a selected range, and then want to read the selected range's address property, the user needs to load the address property before the user can read it. To request properties of a proxy object be loaded, call the load() method on the object and specify the properties to load.en call context.sync().

Just like requests to set properties or invoke methods on proxy objects, requests to load properties on proxy objects get added to the queue of pending commands on the request context, which will run the next time when user call the sync() method. Users can queue up as many loads () calls on the request context as necessary. In the following example, only specific properties of the range are loaded.

```javascript
Excel.run(function (context) {
    var sheetName = 'Sheet1';
    var rangeAddress = 'A1:B2';
    var myRange = context.workbook.worksheets.getItem(sheetName).getRange(rangeAddress);

    myRange.load(['address', 'format/*', 'format/fill', 'entireRow' ]);

    return context.sync()
      .then(function () {
        console.log (myRange.address);              // ok
        console.log (myRange.format.wrapText);      // ok
        console.log (myRange.format.fill.color);    // ok
        //console.log (myRange.format.font.color);  // not ok as it was not loaded
        });
    }).then(function () {
        console.log('done');
}).catch(function (error) {
    console.log('Error: ' + error);
    if (error instanceof OfficeExtension.Error) {
        console.log('Debug info: ' + JSON.stringify(error.debugInfo));
    }
});
```

WORD PACKAGE

Excel.Application	Represents the Excel application that manages the workbook.
Excel.AutoFilter	Represents the AutoFilter object. AutoFilter turns the values in the Excel column into specific filters based on the cell contents.
Excel.Binding	Represents an Office.js binding that is defined in the workbook.
Excel.BindingCollection	Represents the collection of all the binding objects that are part of the workbook.
Excel.CellValueConditionalFormat	Represents a cell value conditional format.
Excel.Chart	Represents a chart object in a workbook. To learn more about the Chart object model, see Work with charts using the Excel JavaScript API.
Excel.ChartAreaFormat	Encapsulates the format properties for the overall chart area.
Excel.ChartAxes	Represents the chart axes.
Excel.ChartAxis	Represents a single axis in a chart.
Excel.ChartAxisFormat	Encapsulates the format properties for the chart axis.
Excel.ChartAxisTitle	Represents the title of a chart axis.
Excel.ChartAxisTitleFormat	Represents the chart axis title formatting.
Excel.ChartBinOptions	Encapsulates the bin options for histogram charts and Pareto charts.
Excel.ChartBorder	Represents the border formatting of a chart element.
Excel.ChartBoxwhiskerOptions	Represents the properties of a box and whisker chart.
Excel.ChartCollection	A collection of all the chart objects on a worksheet.
Excel.ChartDataLabel	Represents the data label of a chart point.
Excel.ChartDataLabelFormat	Encapsulates the format properties for the chart data labels.
Excel.ChartDataLabels	Represents a collection of all the data labels on a chart point.
Excel.ChartErrorBars	This object represents the attributes for a chart's error bars.
Excel.ChartErrorBarsFormat	Encapsulates the format properties for chart error bars.

Excel.ChartFill	Represents the fill formatting for a chart element.
Excel.ChartFont	This object represents the font attributes (font name, font size, color, etc.) for a chart object.
Excel.ChartFormatString	Represents the substring in chart related objects that contain text, like ChartTitle object, ChartAxisTitle object, etc.
Excel.ChartGridlines	Represents major or minor gridlines on a chart axis.
Excel.ChartGridlinesFormat	Encapsulates the format properties for chart gridlines.
Excel.ChartLegend	Represents the legend in a chart.
Excel.ChartLegendEntry	Represents the legendEntry in legendEntryCollection.
Excel.ChartLegendEntryCollection	Represents a collection of legendEntries.
Excel.ChartLegendFormat	Encapsulates the format properties of a chart legend.
Excel.ChartLineFormat	Encapsulates the formatting options for line elements.
Excel.ChartMapOptions	Encapsulates the properties for a region map chart.
Excel.ChartPivotOptions	Encapsulates the options for the pivot chart.
Excel.ChartPlotArea	This object represents the attributes for a chart plotArea object.
Excel.ChartPlotAreaFormat	Represents the format properties for chart plotArea.
Excel.ChartPoint	Represents a point of a series in a chart.
Excel.ChartPointFormat	Represents formatting objects for chart points.
Excel.ChartPointsCollection	A collection of all the chart points within a series inside a chart.
Excel.ChartSeries	Represents a series in a chart.
Excel.ChartSeriesCollection	Represents a collection of chart series.
Excel.ChartSeriesFormat	Encapsulates the format properties for the chart series
Excel.ChartTitle	Represents a chart title object of a chart.
Excel.ChartTitleFormat	Provides access to the office art formatting for the chart title.
Excel.ChartTrendline	This object represents the attributes for a chart trendline object.

Excel.ChartTrendlineCollection	Represents a collection of Chart Trendlines.
Excel.ChartTrendlineFormat	Represents the format properties for chart trendline.
Excel.ChartTrendlineLabel	This object represents the attributes for a chart trendline label object.
Excel.ChartTrendlineLabelFormat	Encapsulates the format properties for the chart trendline label.
Excel.ColorScaleConditionalFormat	Represents ColorScale criteria for conditional formatting.
Excel.Comment	Represents a comment in the workbook.
Excel.CommentCollection	Represents a collection of comment objects that are part of the workbook.
Excel.CommentReply	Represents a comment reply in the workbook.
Excel.CommentReplyCollection	Represents a collection of comment reply objects that are part of the comment.
Excel.ConditionalDataBarNegativeFormat	Represents a conditional format DataBar Format for the negative side of the data bar.
Excel.ConditionalDataBarPositiveFormat	Represents a conditional format DataBar Format for the positive side of the data bar.
Excel.ConditionalFormat	An object encapsulating a conditional format's range, format, rule, and other properties. To learn more about the conditional formatting object model, read Apply conditional formatting to Excel ranges.
Excel.ConditionalFormatCollection	Represents a collection of all the conditional formats that overlap the range.
Excel.ConditionalFormatRule	Represents a rule, for all traditional rule/format pairings.
Excel.ConditionalRangeBorder	Represents the border of an object.
Excel.ConditionalRangeBorderCollection	Represents the border objects that makeup range border.
Excel.ConditionalRangeFill	Represents the background of a conditional range object.
Excel.ConditionalRangeFont	This object represents the font attributes (font style, color, etc.) for an object.
Excel.ConditionalRangeFormat	A formatting object encapsulating the conditional formats range's font, fill borders, and other properties.
Excel.CultureInfo	Provides information based on current system culture settings. This includes the culture names, number formatting, and other culturally dependent settings.
Excel.CustomConditionalFormat	Represents a custom conditional format type.
Excel.CustomProperty	Represents a custom property.

Excel.CustomPropertyCollection	Contains the collection of customProperty objects.
Excel.CustomXmlPart	Represents a custom XML part object in a workbook.
Excel.CustomXmlPartCollection	A collection of custom XML parts.
Excel.CustomXmlPartScopedCollection	A scoped collection of custom XML parts. A scoped collection is the result of some operation (e.g., filtering by namespace). A scoped collection cannot be scoped any further.
Excel.DataBarConditionalFormat	Represents an Excel Conditional Data Bar Type.
Excel.DataConnectionCollection	Represents a collection of all the Data Connections that are part of the workbook or worksheet.
Excel.DataPivotHierarchy	Represents Excel DataPivotHierarchy.
Excel.DataPivotHierarchyCollection	Represents a collection of DataPivotHierarchy items associated with the PivotTable.
Excel.DataValidation	Represents the data validation applied to the current range. To learn more about the data validation object model, read Add data validation to Excel ranges.
Excel.DatetimeFormatInfo	Defines the culturally appropriate format of displaying numbers. This is based on current system culture settings.
Excel.DocumentProperties	Represents workbook properties.
Excel.Filter	Manages the filtering of a table's column.
Excel.FilterPivotHierarchy	Represents Excel FilterPivotHierarchy.
Excel.FilterPivotHierarchyCollection	Represents a collection of FilterPivotHierarchy items associated with the PivotTable.
Excel.FormatProtection	Represents the format protection of a range object.
Excel.FunctionResult	An object containing the result of a function-evaluation operation
Excel.Functions	An object for evaluating Excel functions.
Excel.GeometricShape	Represents a geometric shape inside a worksheet. A geometric shape can be a rectangle, block arrow, equation symbol, flowchart item, star, banner, callout, or any other basic shape in Excel.
Excel.GroupShapeCollection	Represents the shape collection inside a shape group.
Excel.IconSetConditionalFormat	Represents an IconSet criterion for conditional formatting.
Excel.Image	Represents an image in the worksheet. To get the corresponding Shape object, use Image.shape.

Excel.IterativeCalculation	Represents the Iterative Calculation settings.
Excel.Line	Represents a line inside a worksheet. To get the corresponding Shape object, use Line.shape.
Excel.NamedItem	Represents a defined name for a range of cells or values. Names can be primitive named objects (as seen in the type below), range object, or a reference to a range. This object can be used to obtain a range object associated with names.
Excel.NamedItemArrayValues	Represents an object containing values and types of a named item.
Excel.NamedItemCollection	A collection of all the NamedItem objects that are part of the workbook or worksheet, depending on how it was reached.
Excel.NamedSheetView	Represents a named sheet view of a worksheet. A sheet view stores the sort and filter rules for a particular worksheet. Every sheet view (even a temporary sheet view) has a unique, worksheet-scoped name that is used to access the view.
Excel.NamedSheetViewCollection	Represents the collection of sheet views on the worksheet.
Excel.NumberFormatInfo	Defines the culturally appropriate format of displaying numbers. This is based on current system culture settings.
Excel.PageLayout	Represents layout and print settings that are not dependent on any printer-specific implementation. These settings include margins, orientation, page numbering, title rows, and print area.
Excel.PivotField	Represents Excel PivotField.
Excel.PivotFieldCollection	Represents a collection of all the PivotFields that are part of a PivotTable's hierarchy.
Excel.PivotHierarchy	Represents Excel PivotHierarchy.
Excel.PivotHierarchyCollection	Represents a collection of all the PivotHierarchies that are part of the PivotTable.
Excel.PivotItem	Represents Excel PivotItem.
Excel.PivotItemCollection	Represents a collection of all the PivotItems related to their parent PivotField.
Excel.PivotLayout	Represents the visual layout of the PivotTable.
Excel.PivotTable	Represents an Excel PivotTable. To learn more about the PivotTable object model, read Work with PivotTables using the Excel JavaScript API.
Excel.PivotTableCollection	Represents a collection of all the PivotTables that are part of the workbook or worksheet.
Excel.PivotTableScopedCollection	Represents a scoped collection of PivotTables. The PivotTables are sorted based on the location of the PivotTable's top-left corner. They are ordered top to bottom and then left to right.

Excel.PivotTableStyle	Represents a PivotTable Style, which defines style elements by the PivotTable region.
Excel.PivotTableStyleCollection	Represents a collection of PivotTable styles.
Excel.PresetCriteriaConditionalFormat	Represents the preset criteria conditional format such as above average, below average, unique values contains a blank, nonblank, error, and no error.
Excel.Range	The range represents a set of one or more contiguous cells such as a cell, a row, a column, a block of cells, etc. To learn more about how ranges are used throughout the API, read Work with ranges using the Excel JavaScript API and Work with ranges using the Excel JavaScript API (advanced).
Excel.RangeAreas	RangeAreas represents a collection of one or more rectangular ranges in the same worksheet. To learn how to use discontiguous ranges, read Work with multiple ranges simultaneously in Excel add-ins.
Excel.RangeAreasCollection	Contains the collection of cross-worksheets level Ranges.
Excel.RangeBorder	Represents the border of an object.
Excel.RangeBorderCollection	Represents the border objects that make up the range border.
Excel.RangeFill	Represents the background of a range object.
Excel.RangeFont	This object represents the font attributes (font name, font size, color, etc.) for an object.
Excel.RangeFormat	A formatting object encapsulating the range's font, fill borders, alignment, and other properties.
Excel.RangeSort	Manages sorting operations on Range objects.
Excel.RangeView	RangeView represents a set of visible cells of the parent range.
Excel.RangeViewCollection	Represents a collection of RangeView objects.
Excel.RemoveDuplicatesResult	Represents the results from the removeDuplicates method on a range
Excel.RequestContext	The RequestContext object facilitates requests to the Excel application. Since the Office add-in and the Excel application run in two different processes, the request context is required to get access to the Excel object model from the add-in.
Excel.RowColumnPivotHierarchy	Represents Excel RowColumnPivotHierarchy.
Excel.RowColumnPivotHierarchyCollection	Represents a collection of RowColumnPivotHierarchy items associated with the PivotTable.
Excel.Runtime	Represents the Excel Runtime class.

Excel.Session	Provides a connection session for a remote workbook.
Excel.Setting	The setting represents a key-value pair of a setting persisted to the document (per file per add-in). These custom key-value pairs can be used to store state or lifecycle information needed by the content or task-pane add-in. Note that settings are persisted in the document and hence it is not a place to store any sensitive or protected information such as user information and password.
Excel.SettingCollection	Represents a collection of key-value pair setting objects that are part of the workbook. The scope is limited to per file and add-in (task-pane or content) combination.
Excel.Shape	Represents a generic shape object in the worksheet. A shape could be a geometric shape, a line, a group of shapes, etc. To learn more about the shape object model, read Work with shapes using the Excel JavaScript API.
Excel.ShapeCollection	Represents a collection of all the shapes in the worksheet.
Excel.ShapeFill	Represents the fill formatting of a shape object.
Excel.ShapeFont	Represents the font attributes, such as font name, font size, and color, for a shape's TextRange object.
Excel.ShapeGroup	Represents a shape group inside a worksheet. To get the corresponding Shape object, use ShapeGroup.shape.
Excel.ShapeLineFormat	Represents the line formatting for the shape object. For images and geometric shapes, line formatting represents the border of the shape.
Excel.Slicer	Represents a slicer object in the workbook.
Excel.SlicerCollection	Represents a collection of all the slicer objects on the workbook or a worksheet.
Excel.SlicerItem	Represents a slicer item in a slicer.
Excel.SlicerItemCollection	Represents a collection of all the slicer item objects on the slicer.
Excel.SlicerStyle	Represents a Slicer Style, which defines style elements by region of the slicer.
Excel.SlicerStyleCollection	Represents a collection of SlicerStyle objects.
Excel.Style	An object encapsulating a style's format and other properties.
Excel.StyleCollection	Represents a collection of all the styles.
Excel.Table	Represents an Excel table. To learn more about the table object model, read Work with tables using the Excel JavaScript API.
Excel.TableCollection	Represents a collection of all the tables that are part of the workbook or worksheet, depending on how it was reached.

Excel.TableColumn	Represents a column in a table.
Excel.TableColumnCollection	Represents a collection of all the columns that are part of the table.
Excel.TableRow	Represents a row in a table. Note that unlike Ranges or Columns, which will adjust if new rows/columns are added before them, a TableRow object represents the physical location of the table row, but not the data. That is, if the data is sorted or if new rows are added, a table row will continue to point at the index for which it was created.
Excel.TableRowCollection	Represents a collection of all the rows that are part of the table. Note that unlike Ranges or Columns, which will adjust if new rows/columns are added before them, a TableRow object represents the physical location of the table row, but not the data. That is, if the data is sorted or if new rows are added, a table row will continue to point at the index for which it was created.
Excel.TableScopedCollection	Represents a scoped collection of tables. For each table, its top-left corner is considered its anchor location, and the tables are sorted top to bottom and then left to right.
Excel.TableSort	Manages sorting operations on Table objects.
Excel.TableStyle	Represents a TableStyle, which defines the style elements by region of the Table.
Excel.TableStyleCollection	Represents a collection of TableStyles.
Excel.TextConditionalFormat	Represents a specific text conditional format.
Excel.TextFrame	Represents the text frame of a shape object.
Excel.TextRange	Contains the text that is attached to a shape, in addition to properties and methods for manipulating the text.
Excel.TimelineStyle	Represents a Timeline style, which defines style elements by region in the Timeline.
Excel.TimelineStyleCollection	Represents a collection of TimelineStyles.
Excel.TopBottomConditionalFormat	Represents a Top/Bottom conditional format.
Excel.Workbook	The workbook is the top-level object which contains related workbook objects such as worksheets, tables, ranges, etc. To learn more about the workbook object model, read Work with workbooks using the Excel JavaScript API.
Excel.WorkbookCreated	The WorkbookCreated object is the top-level object created by Application.CreateWorkbook. A WorkbookCreated object is a special Workbook object.

Excel.WorkbookProtection	Represents the protection of a workbook object.
Excel.WorkbookRangeAreas	Represents a collection of one or more rectangular ranges in multiple worksheets.
Excel.Worksheet	An Excel worksheet is a grid of cells. It can contain data, tables, charts, etc. To learn more about the worksheet object model, read Work with worksheets using the Excel JavaScript API.
Excel.WorksheetCollection	Represents a collection of worksheet objects that are part of the workbook.
Excel.WorksheetCustomProperty	Represents a worksheet-level custom property.
Excel.WorksheetCustomPropertyCollection	Contains the collection of the worksheet-level custom property.
Excel.WorksheetProtection	Represents the protection of a sheet object.

FUNCTIONS

Excel.createWorkbook(base64)	**Creates and opens a new workbook. Optionally, the workbook can be pre-populated with a base64-encoded .xlsx file.**
Excel.run(batch)	Executes a batch script that performs actions on the Excel object model, using a new RequestContext. When the promise is resolved, any tracked objects that were automatically allocated during execution will be released.
Excel.run(object, batch)	Executes a batch script that performs actions on the Excel object model, using the RequestContext of a previously-created API object. When the promise is resolved, any tracked objects that were automatically allocated during execution will be released.
Excel.run(objects, batch)	Executes a batch script that performs actions on the Excel object model, using the RequestContext of previously-created API objects.
Excel.run(options, batch)	Executes a batch script that performs actions on the Excel object model, using the RequestContext of a previously-created API object. When the promise is resolved, any tracked objects that were automatically allocated during execution will be released.
Excel.run(context, batch)	Executes a batch script that performs actions on the Excel object model, using the RequestContext of a previously-created object. When the promise is resolved, any tracked objects that were automatically allocated during execution will be released.

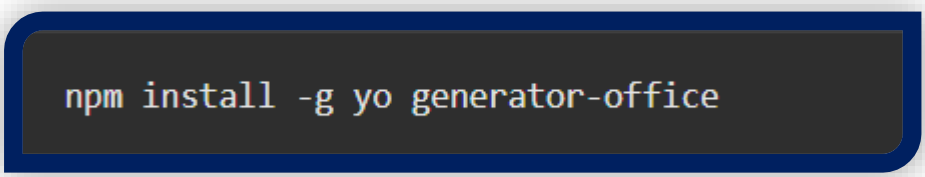
CREATE THE ADD-IN

Users can create an Office Add-in by using the Yeoman generator for Office Add-ins or Visual Studio. The Yeoman generator creates a Node.js project that can be managed with Visual Studio Code or any other editor, whereas Visual Studio creates a Visual Studio solution.

METHOD 1 - OFFICE 365 EXCEL WEB ADDIN - YEOMAN GENERATOR - VISUAL CODE

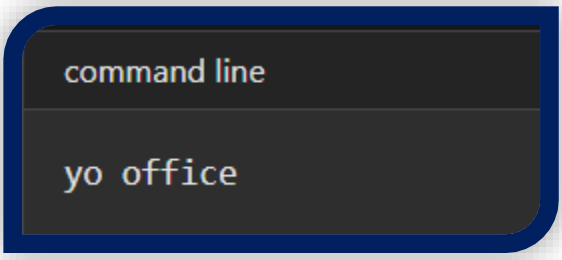
Prerequisites

1. Node.js (the latest LTS version)
2. The latest version of Yeoman and the Yeoman generator for Office Add-ins. To install these tools globally, run the following command via the command prompt:



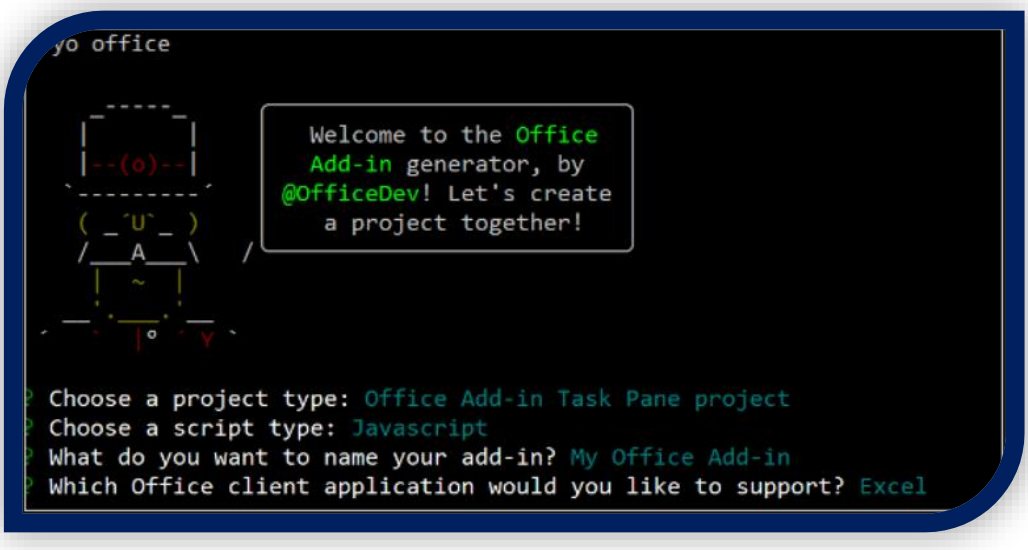
Create the add-in project

1. Run the following command to create an add-in project using the Yeoman generator:



2. When prompted, provide the following information to create your add-in project:
   - **Choose a project type:** Office Add-in Task Pane project
   - **Choose a script type**: Javascript

- **What do you want to name your add-in?** My Office Add-in
- **Which Office client application would you like to support?** Excel



After completing the wizard, the generator creates the project and installs supporting Node components.

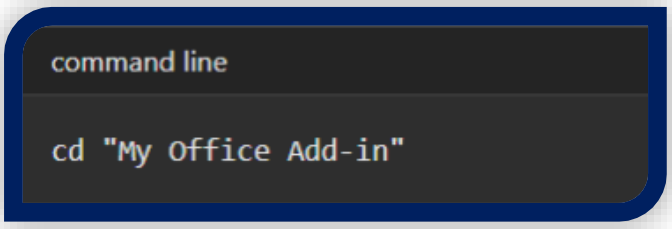
Explore the project

The add-in project that the user has just created with the Yeoman generator contains sample code for a very basic task pane add-in. To explore the components of the add-in project, open the project in the code editor, and review the files listed below.

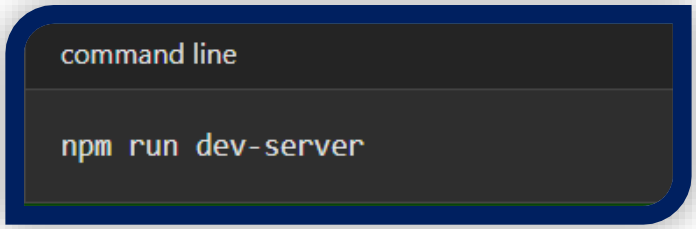
- The ./manifest.xml file in the root directory of the project defines the settings and capabilities of the add-in.
- The ./src/taskpane/taskpane.html file contains the HTML markup for the task pane.
- The ./src/taskpane/taskpane.css file contains the CSS that's applied to content in the task pane.
- The ./src/taskpane/taskpane.js file contains the Office JavaScript API code that facilitates interaction between the task pane and the Office host application.

Try it out

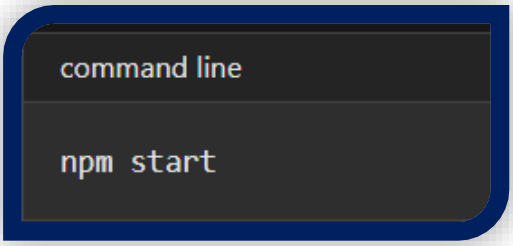
1. Navigate to the root folder of the project.



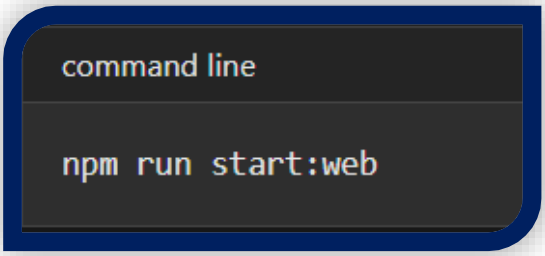
2. Complete the following steps to start the local web server and sideload your add-in.



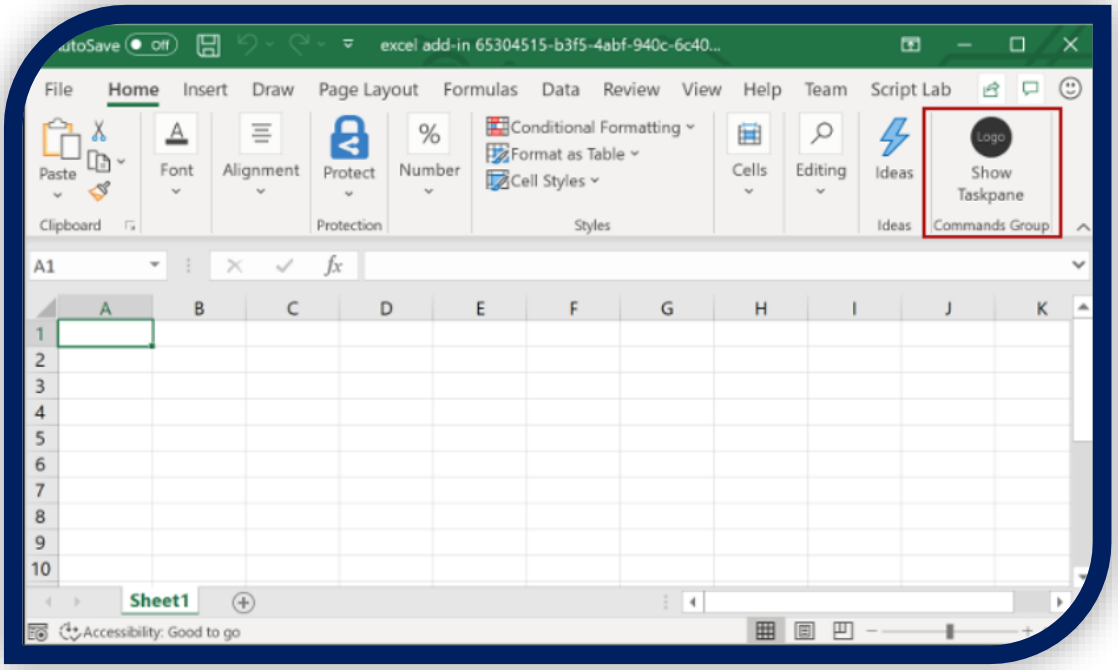
a. To test the add-in in Excel, run the following command in the root directory of the project. This starts the local webserver (if it's not already running) and opens Word with the add-in loaded.



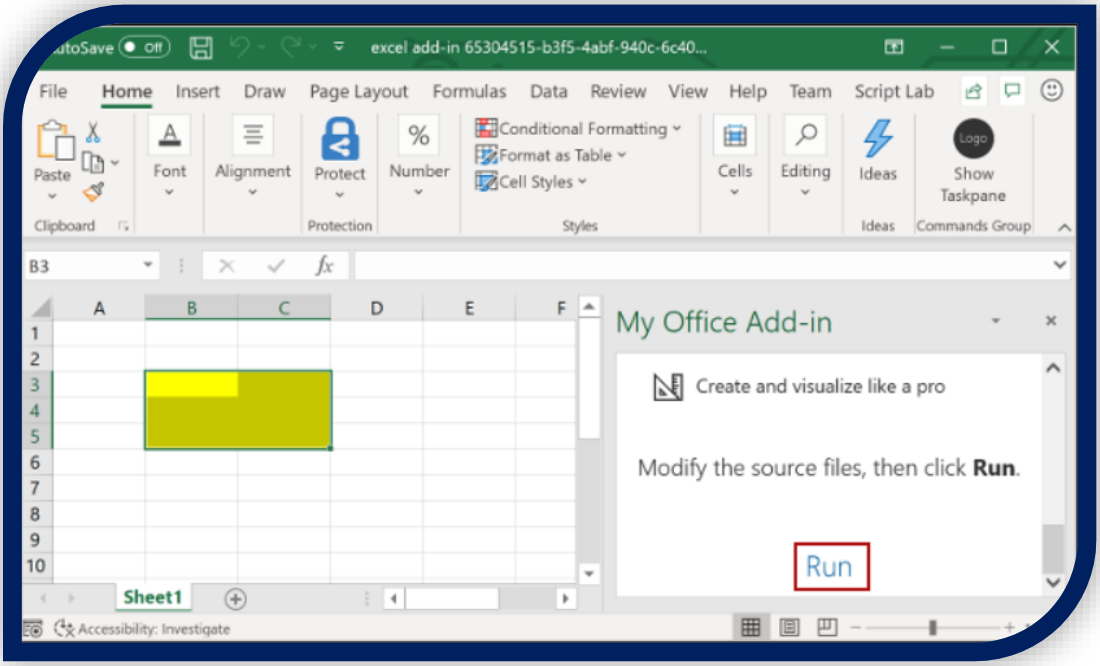
b. To test the add-in in Excel on a browser, run the following command in the root directory of the project. When this command runs, the local web server will start (if it's not already running).



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



4. Select any range of cells in the worksheet.
5. At the bottom of the task pane, choose the Run link to set the color of the selected range to yellow.



METHOD 2 - OFFICE 365 EXCEL WEB ADDIN - VISUAL STUDIO

Prerequisites

a. Visual Studio 2019 with the Office/SharePoint development workload installed
b. Office 2016 or later

Create the add-in project

1. In Visual Studio, choose to Create a new project.
2. Using the search box, enter add-in. Choose Excel Web Add-in, then select Next.
3. Name your project and select Create.
4. In the Create Office Add-in dialog window, choose to 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.

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.

```
<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 at 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').click(setColor);
        });
    });

    function setColor() {
        Excel.run(function (context) {
            var range = context.workbook.getSelectedRange();
            range.format.fill.color = 'green';

            return 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 at 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;
}
```

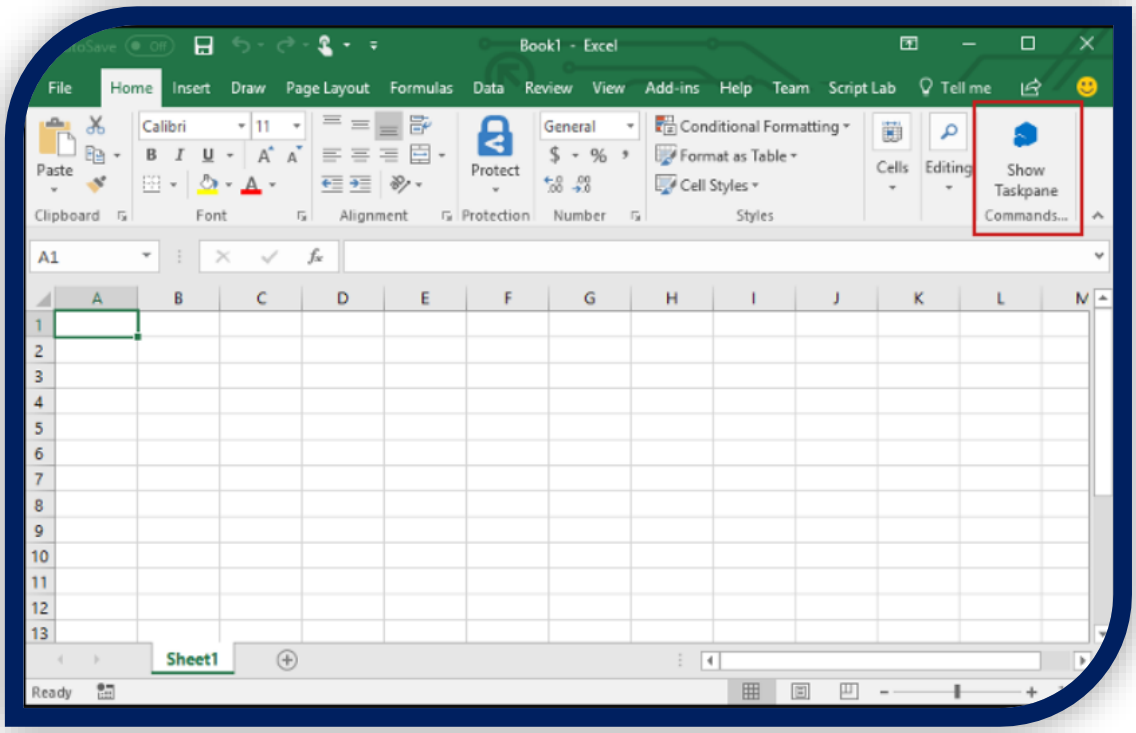
Update the manifest

1. Open the XML manifest file in the add-in project. This file defines the add-in's settings and capabilities.
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.

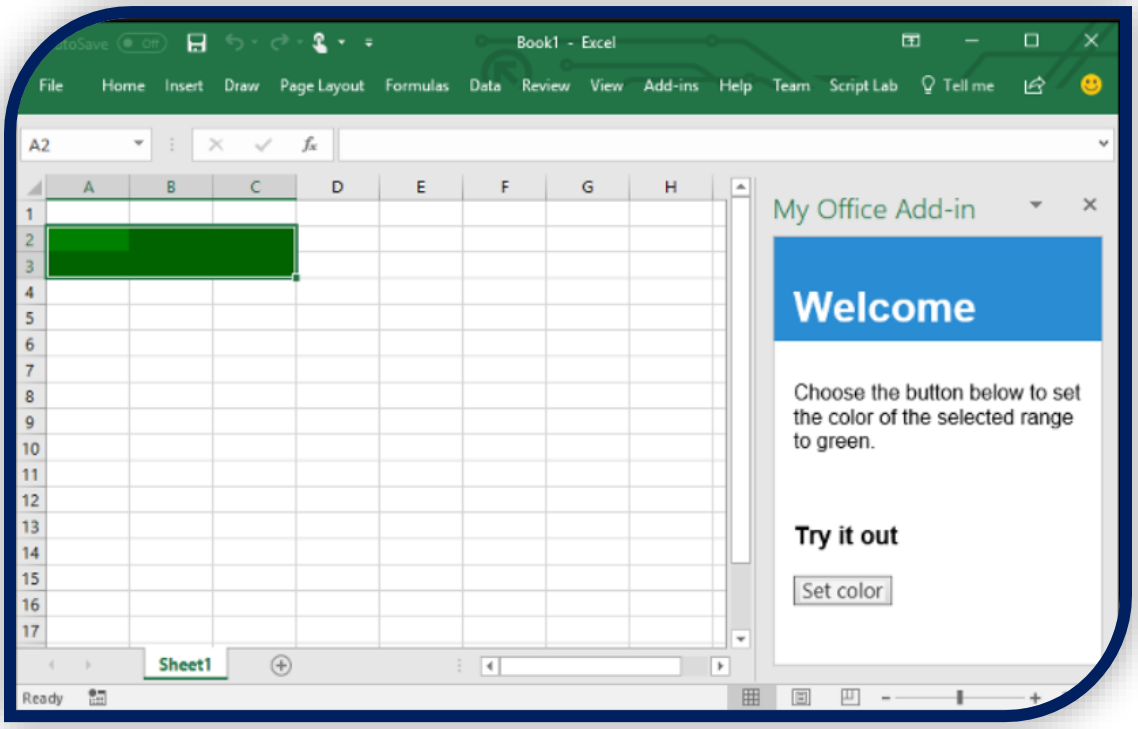
```
<ProviderName>John Doe</ProviderName>
<DefaultLocale>en-US</DefaultLocale>
<!-- The display name of your add-in. Used on the store and various places of the Office UI such as the add-ins dialog. -->
<DisplayName DefaultValue="My Office Add-in" />
<Description DefaultValue="A task pane add-in for 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 in the ribbon. The add-in will be hosted locally on IIS.
2. In Excel, choose the **Home** tab, and then choose the Show Taskpane button in 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.



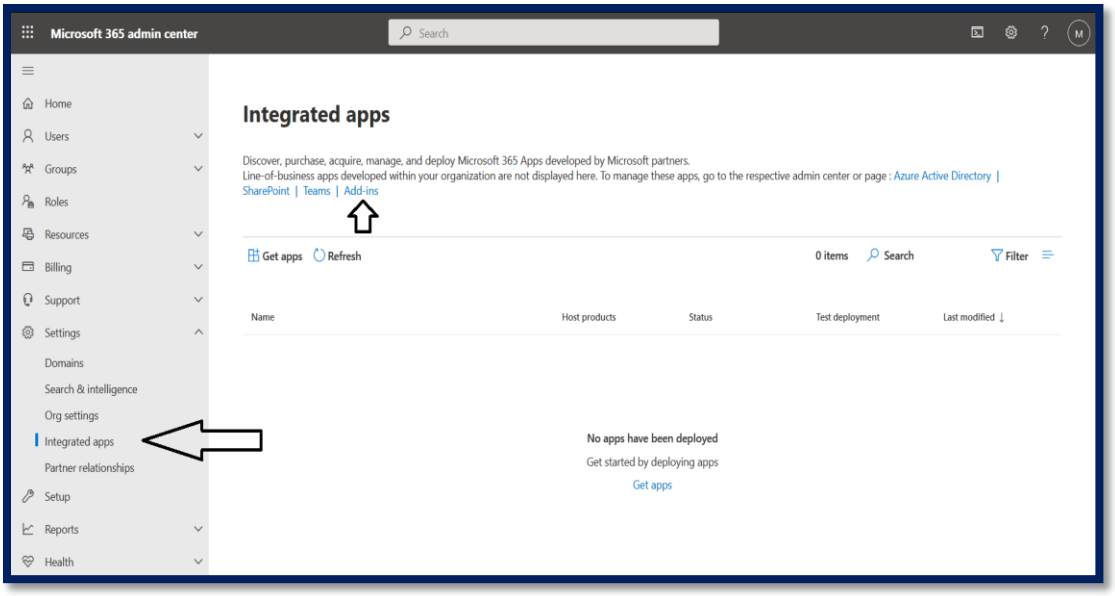
DEPLOYMENT OF ADDIN

There are multiple methods of deployment but we have followed the Centralized Deployment.

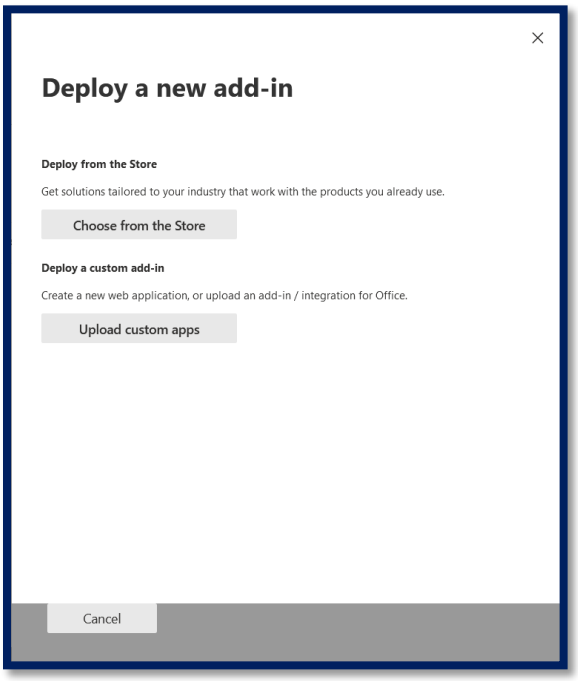
CENTRALIZED DEPLOYMENT

Follow steps below to publish an Office Add-in via Centralized Deployment:

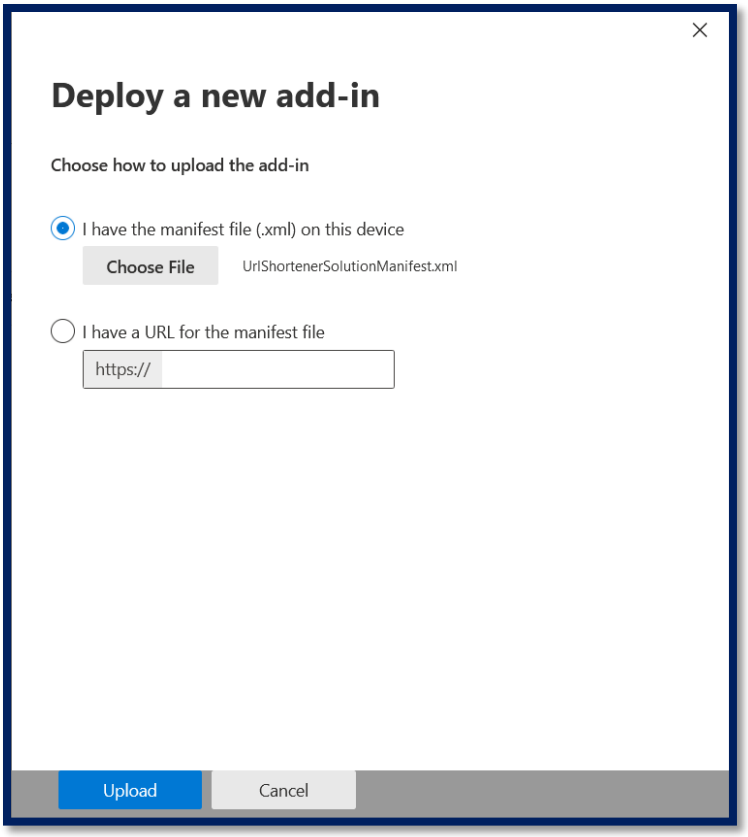
1. Sign in to Microsoft 365 with your work or education account.
2. Select the app launcher icon in the upper-left and choose **Admin**.
3. In the navigation menu, press **Show more**, then choose **Settings** > **Integrated apps**.
4. Choose **Add-ins** at the top of the page



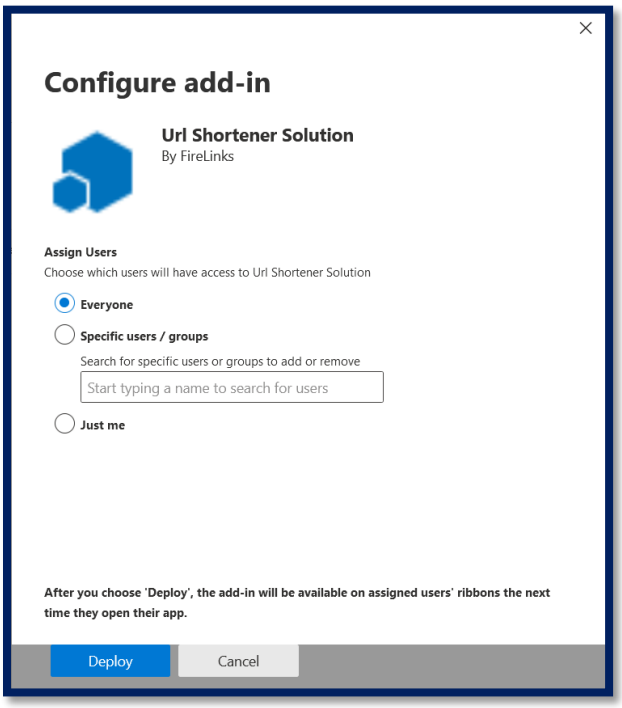
5. Choose **Deploy Add-In** at the top of the page.
6. Choose **Next** after reviewing the requirements.
7. Choose **Upload Custom apps** in the following page.



8. Click on **Choose file** and select Add-in manifest file (.xml)
9. Choose **Upload** at the end of the task pane



10. On the **Assign Users** page, choose **Everyone**, **Specific Users/Groups**, or **Only me**. Use the search box to find the users and groups to whom you want to deploy the add-in.

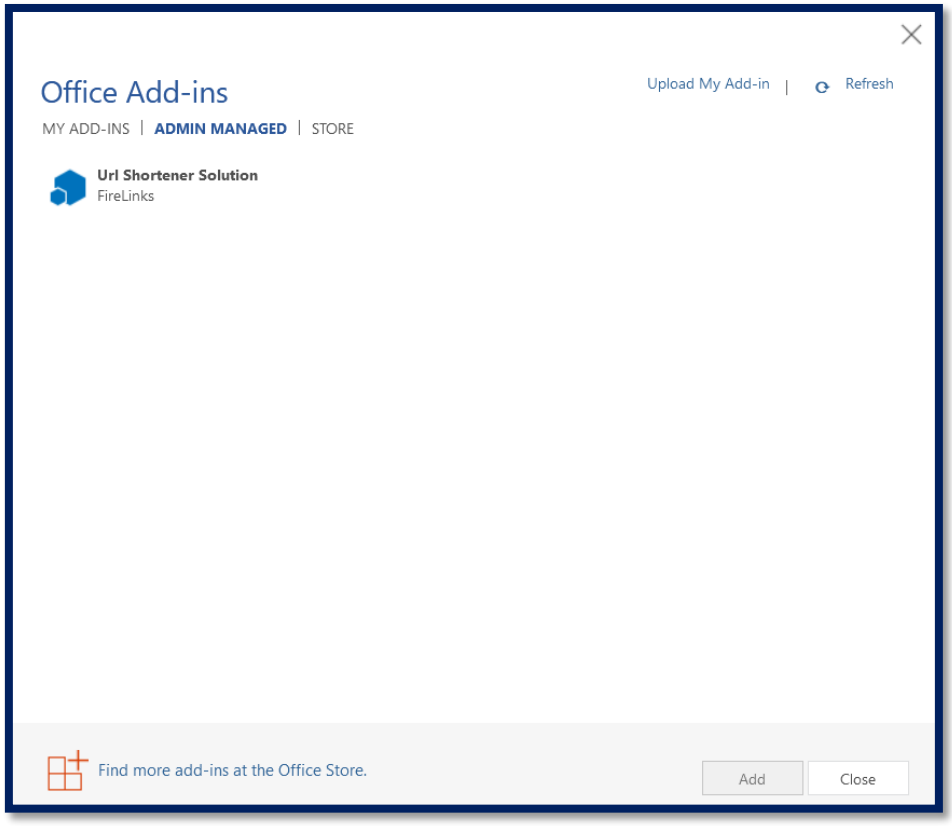


11. When finished, choose **Deploy**. This process may take up to three minutes. Then, finish the walkthrough by pressing **Next**.

DEPLOY ON WEB

Now the add-in has been deployed in your tenant and you can test it on Excel web application following steps below:

1) Open a **new blank document** in Excel web application
2) Go to **Insert** tab
3) Choose **Office Add-ins** in top navigation
4) Choose **Admin managed** in the top of Office Add-ins modal
5) Select your deployed add-in from the list
6) Choose **Add** at the end of the modal



DEPLOY ON DESKTOP

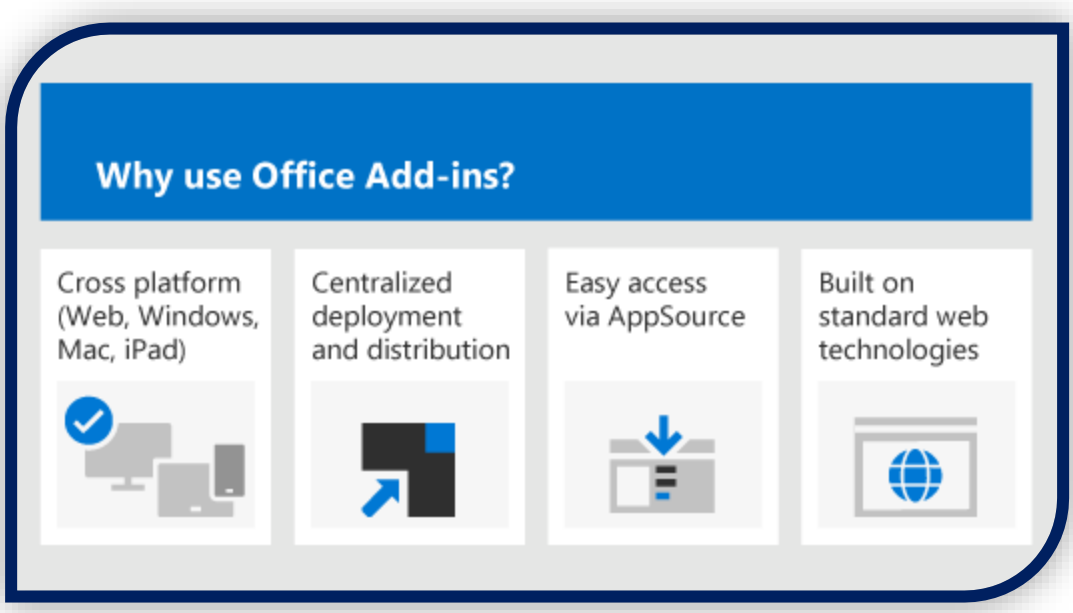
You can test the solution on desktop Excel application by following the steps below:

1) Open a **new blank document** in desktop Excel application
2) Select **File** > **Account**
3) If you're not already signed in, click **Sign In**, else choose **Switch account**
4) Choose **Sign-in with a different account**

5) Provide **login** credentials of Office account on which add-in has been deployed in Office 365
6) Follow the steps 2 to 6 from heading "Test on Web"

HOW ARE OFFICE ADD-INS DIFFERENT FROM COM AND VSTO ADD-INS?

COM or VSTO add-ins are earlier Office integration solutions that run only on Office on Windows. Unlike COM add-ins, Office Add-ins don't involve code that runs on the user's device or in the Office client. For an Office Add-in, the host application, for example, Excel, reads the add-in manifest and hooks up the add-in's custom ribbon buttons and menu commands in the UI. When needed, it loads the add-in's JavaScript and HTML code, which executes in the context of a browser in a sandbox.



Office Add-ins provide the following advantages over add-ins built using VBA, COM, or VSTO:

- **Cross-platform support**. Office Add-ins run in Office on the web, Windows, Mac, and iPad.
- **Centralized deployment and distribution**. Admins can deploy Office Add-ins centrally across an organization.
- **Easy access via AppSource**. You can make your solution available to a broad audience by submitting it to AppSource.
- **Based on standard web technology**. You can use any library you like to build Office Add-ins.

EXAMPLES OF EXCEL WEB ADD-IN IN APPSOURCE

Excel Importer

Company: Theta Systems Limited

App Source: URL

Description: Features and benefits of using this extension

- Import journals, sales and purchase orders/invoices, sales and vendor prices.
- Import Item Journals with item tracking information (lot and serial number)
- Flexible field mapping to suite column layouts of source data.
- Ability to apply default values to fields which are not present in the source data. For example, defaulting a quantity or defaulting dimensions specific to the import.
- Ability to translate source data values to correct accounts and dimensions in Business Central. Examples include:
    - Text string to a combination of accounts and dimensions and VAT/GST posting groups.
    - Old G/L Account, Customer or Vendor numbers to new numbers.

Intrinio Screener for Excel

Company: Intrinio

App Source: URL

Description: Intrinio Screener for Excel allows you to filter all United States equities on more than 500 parameters to find stocks that meet your search criteria.

Screen based on:

- Current Stock Price
- Price to Earnings Ratio
- Return on Equity
- Dividend Yield
- Any item on a 10Q or 10k
- Earnings Per Share
- Sector
- Employee Count

- And more than 500 more parameters…

The resulting list of stocks can be easily integrated with Intrinio's free Excel add-in to analyze the data in a watch list, DCF, or custom model. To use the screener you must visit Intrinio's dashboard (https://data.intrinio.com) to obtain your free API username and password that will allow you to access the screener and Excel Add-In.

Facebook Ads Manager for Excel

Company: Facebook Inc

App Source: URL

Description: Facebook Ads Manager for Excel connects the power of Excel to your Facebook ad accounts.

Excel is a critical tool to analyze ad performance data, but exporting multiple Facebook accounts to an Excel worksheet takes time and effort. With Facebook Ads Manager for Excel you can quickly run a single report to download data from multiple ad accounts, helping you save time and work faster.

With Facebook Ads Manager for Excel users can:

1. Download rich performance data from your ad accounts. You can report on the performance of your campaigns, ad sets, or ads and see breakdowns by demographics, actions taken on ads, or time. You can also filter data based on date ranges, delivery status, ad objectives, placements and more.
2. Create a single report to download your ad performance data from multiple ad accounts. You don't need to export reports from each of your ad accounts in Power Editor, and you don't need to be a developer or work with a Facebook Marketing partner.
3. Create and save custom templates so that you can quickly run the reports you need. Or, use one of our pre-set templates to get started.
4. Refresh the data in your reports as often as you need to, whether that's once a week or every 15 minutes.
5. Take your ad performance data and create your own analysis using Excel's pivot tables or other tools.

CONCLUSION

In this case study, a brief introduction about the Microsoft Office 365 Excel Web add-in, its architecture, development & deployment ways & a demo of the introductory level is discussed in detail.

Our Microsoft Office 365 Consulting, add-in Development, Customization, Integration services, and solutions, can help companies maximize business performance, overcoming market challenges, achieving profitability, and providing the best customer care service.

CONTACT US

Shahzad Sarwar

Cognitive Convergence Team