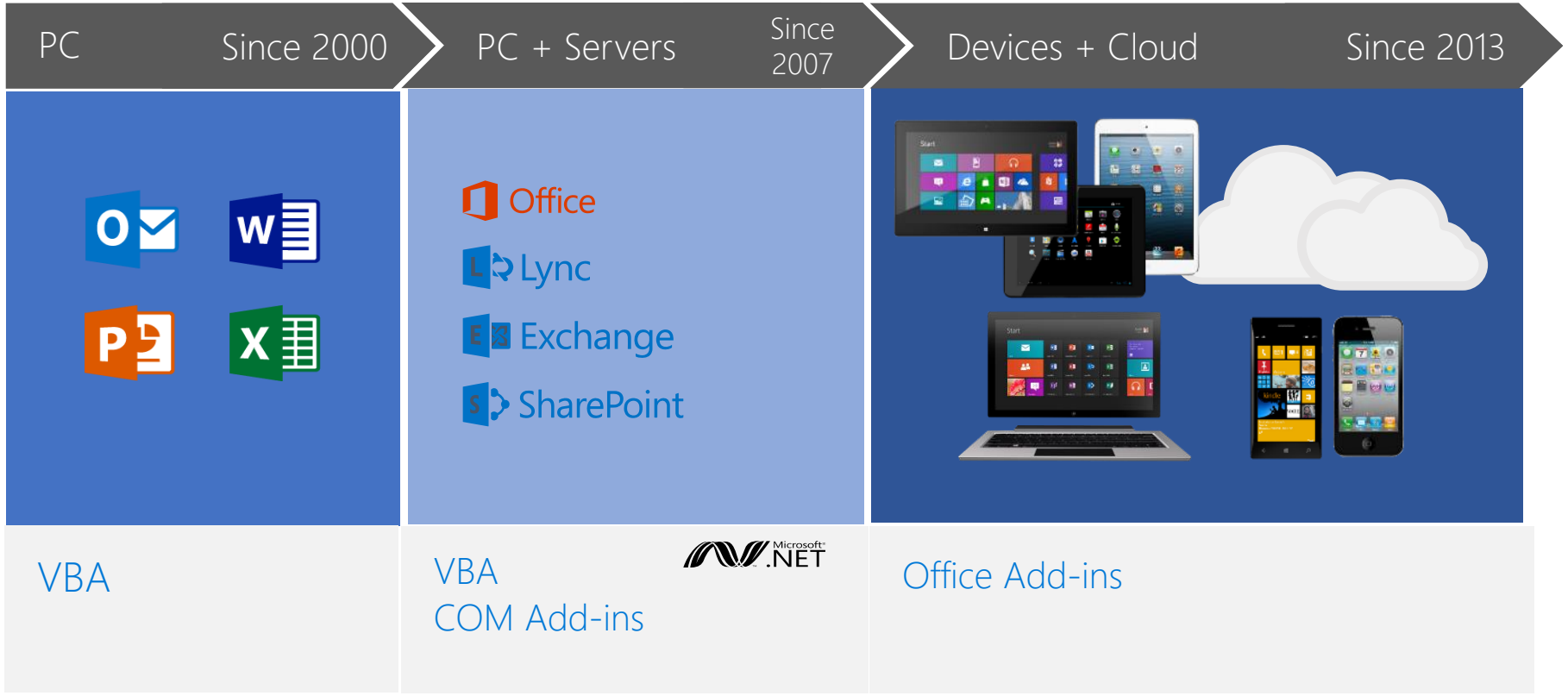
Script Lab Workshop

Office Add-ins History



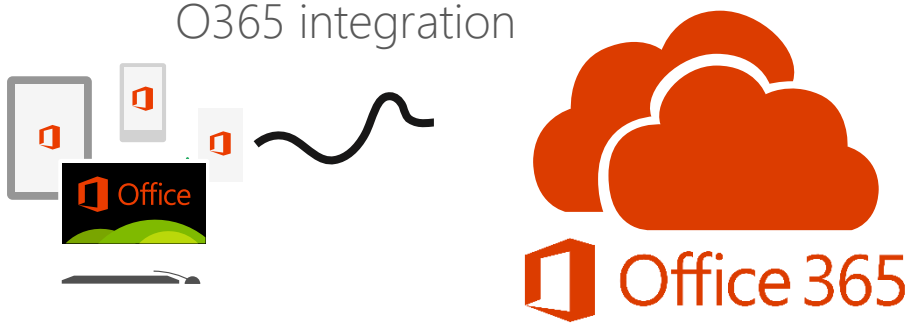
Why Office Add-ins?

Build once, run everywhere

Streamlined lifecycle

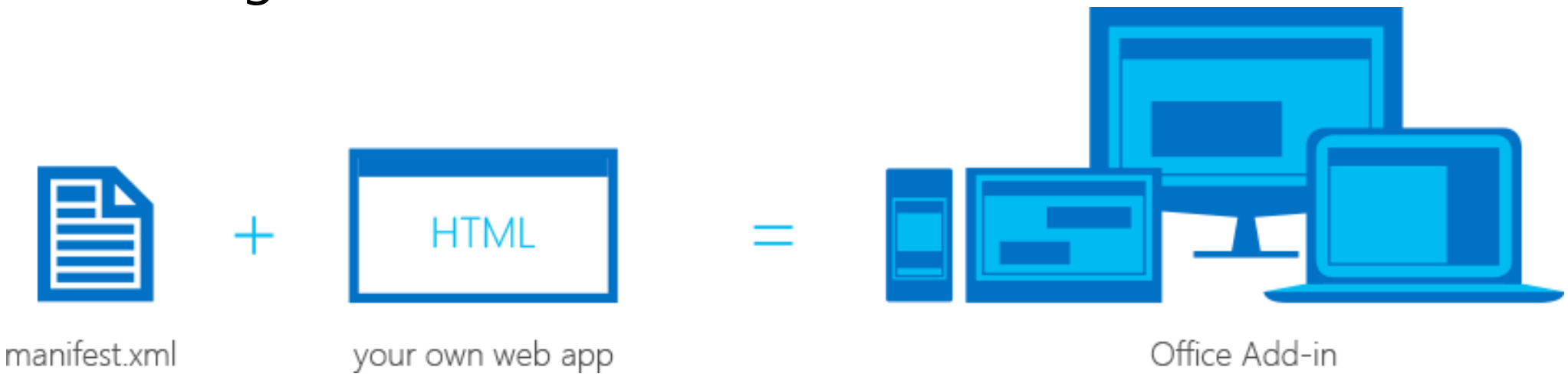
Web standards, open platform

O365 integration



Office Add-ins

Extend Office clients across platforms using web technologies



npm install –g yo generator-office

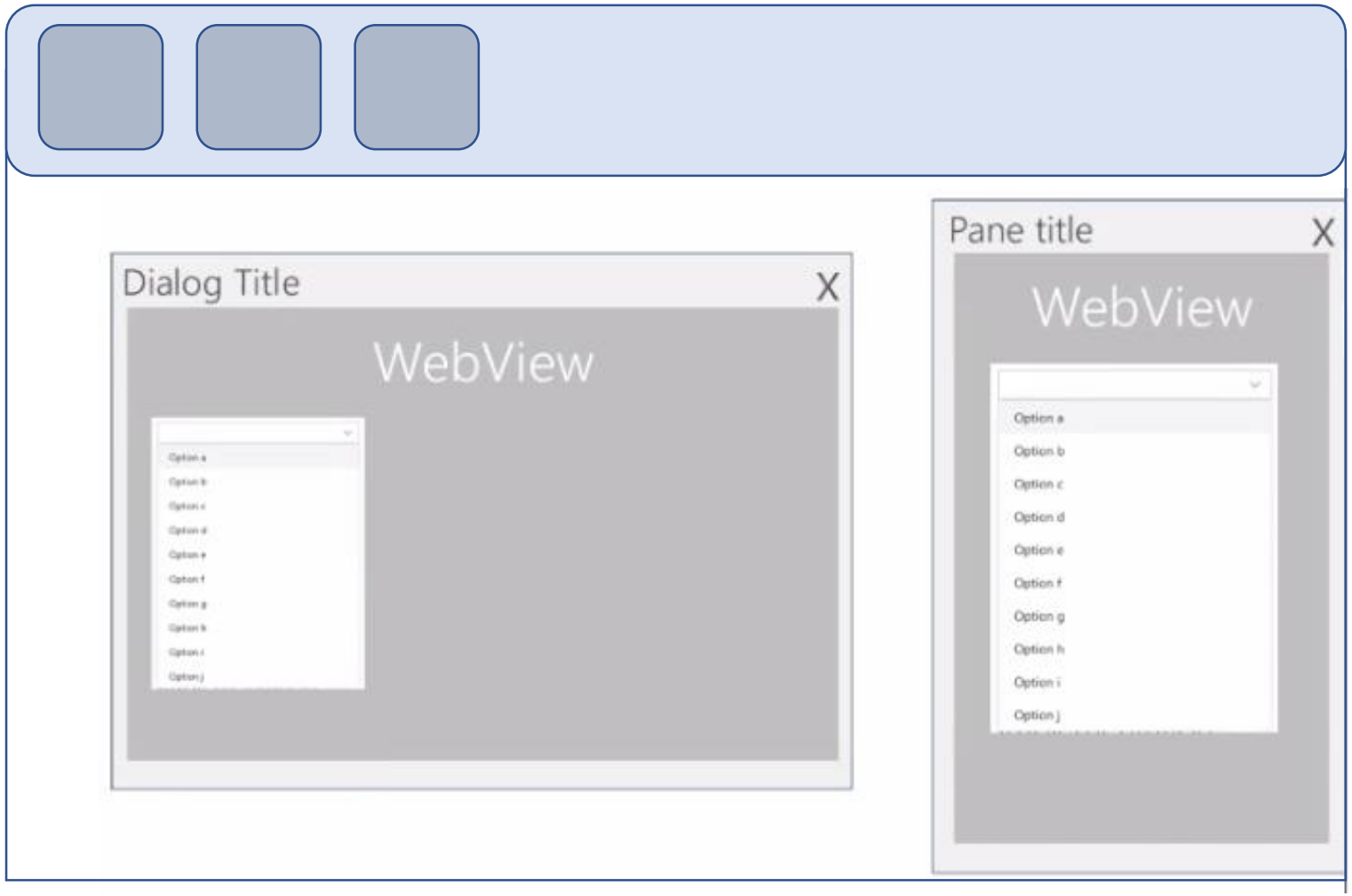
https://dev.office.com/getting-started/addins

+ your favorite IDE/Editor

Visual Studio

Free Visual Studio 2015/2017 Community Edition

Office Add-ins User Interface Elements



Add-in Commands
(e.g. Ribbon Buttons)

HTML Canvases
(e.g. Taskpanes, Dialogs)

Office UI Fabric (optional)
(e.g. dropdown controls)

Add-In Commands

Custom UI hooks into Office clients

Entry points

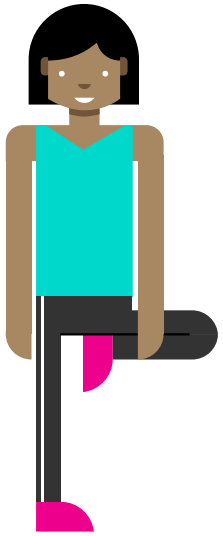
- Buttons on existing tabs
- Buttons on a custom tab
- Contextual Menus

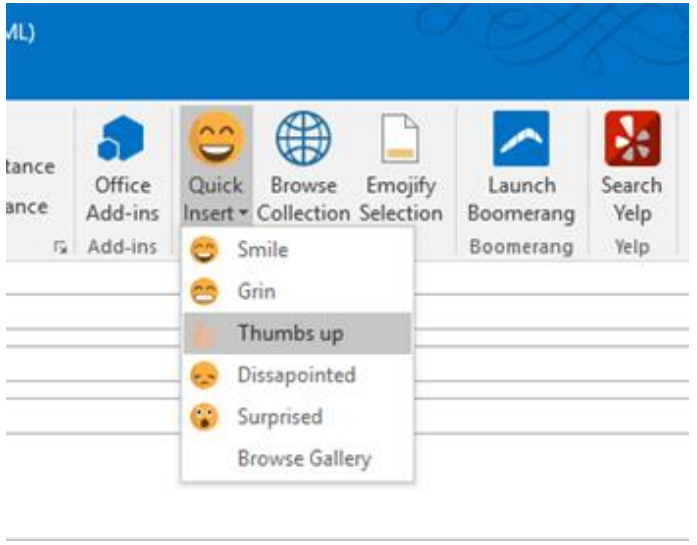
Actions

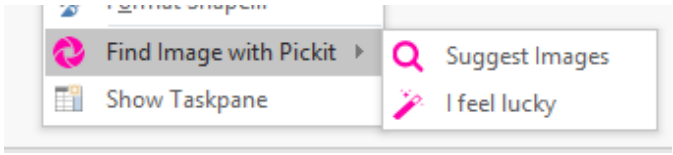
- ShowTaskpane
- ExecuteFunction

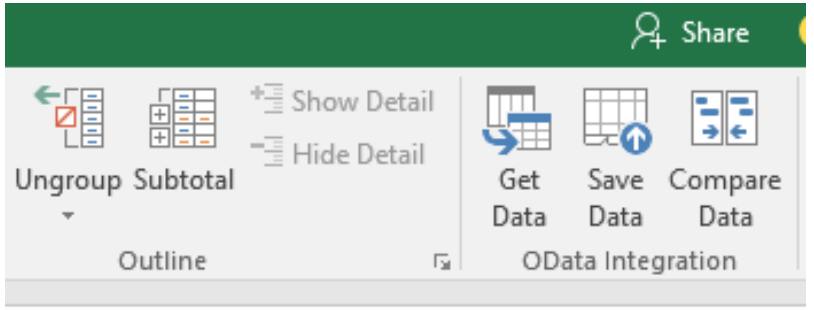
InfoBar

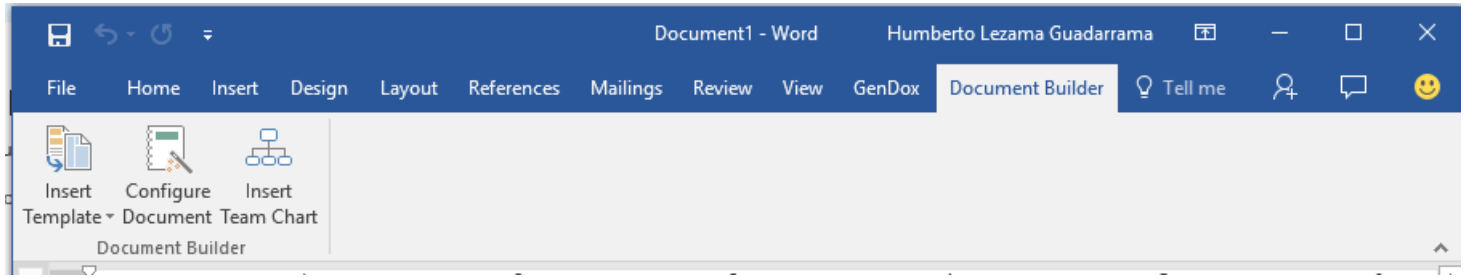
- Show simple text msgs
- Alerts/dialogs not allowed











Add-in commands on the Mac

Ribbon and context menus

Show a pane or execute silent function on command activation

Exactly same manifest and code:

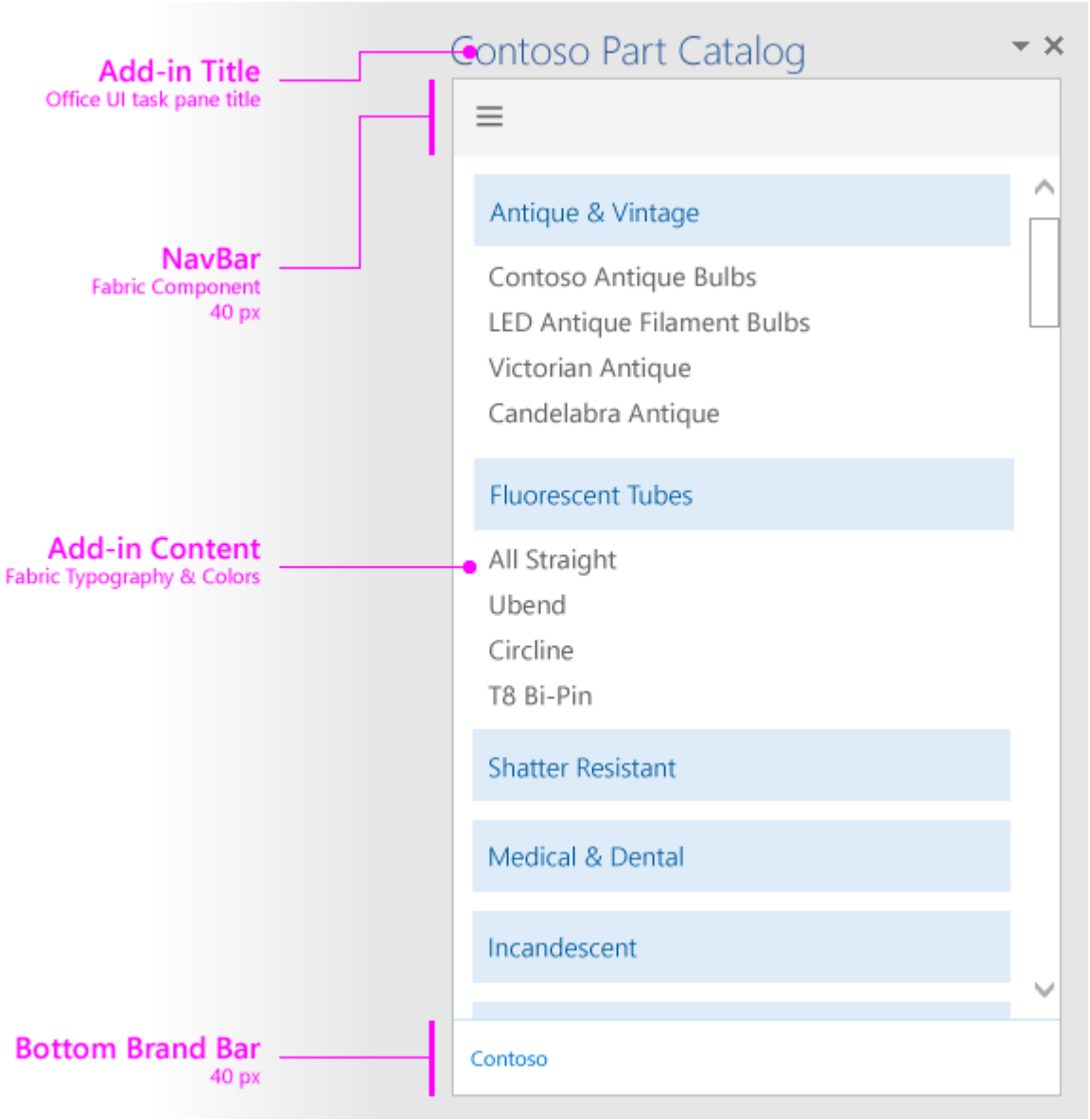
- Office for Windows Desktop
- Office Online
- **Now for Office Mac**

Task Pane Layout Recommendation

Navigation element (optional) <= 80 pixels.

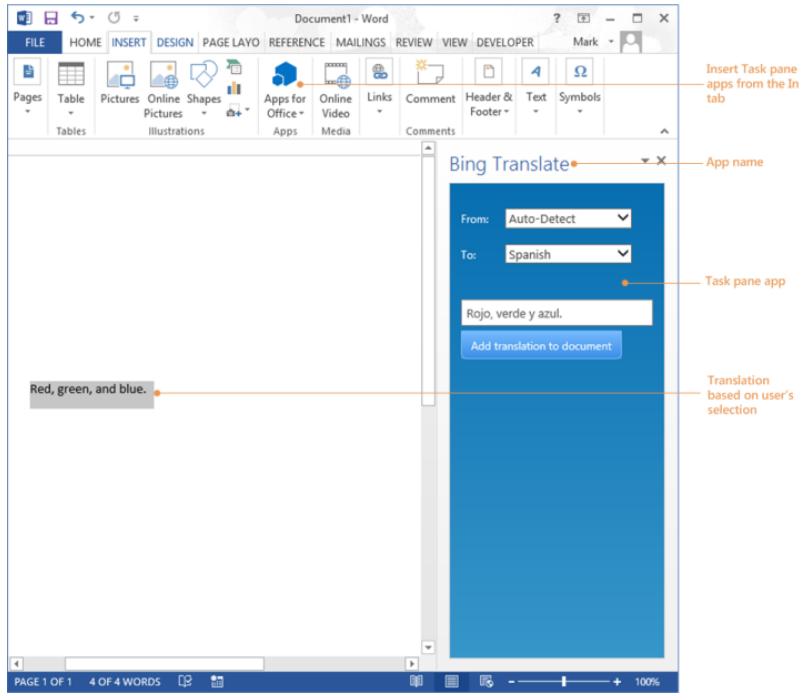
Add-in content

Branding element (optional) 40 pixels

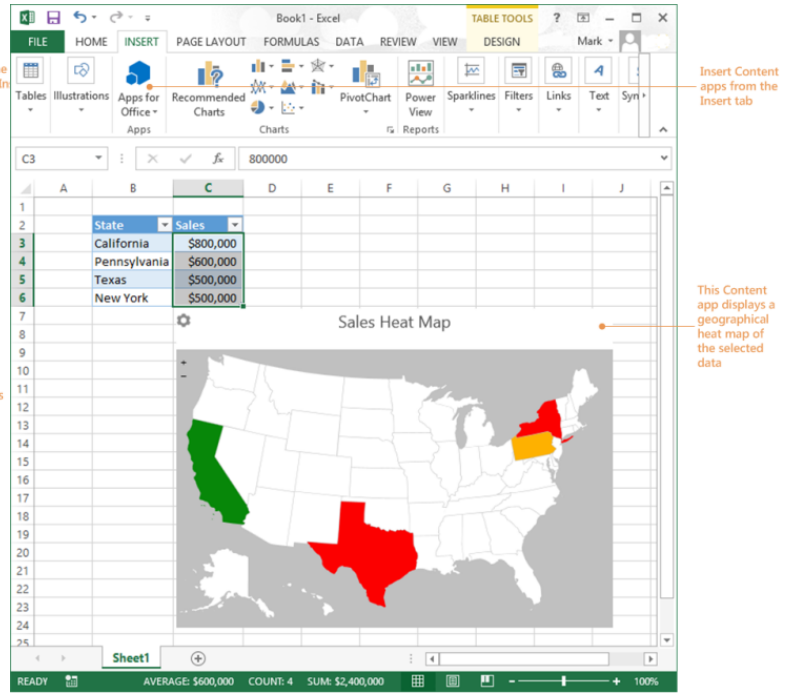


Types of Office Add-ins

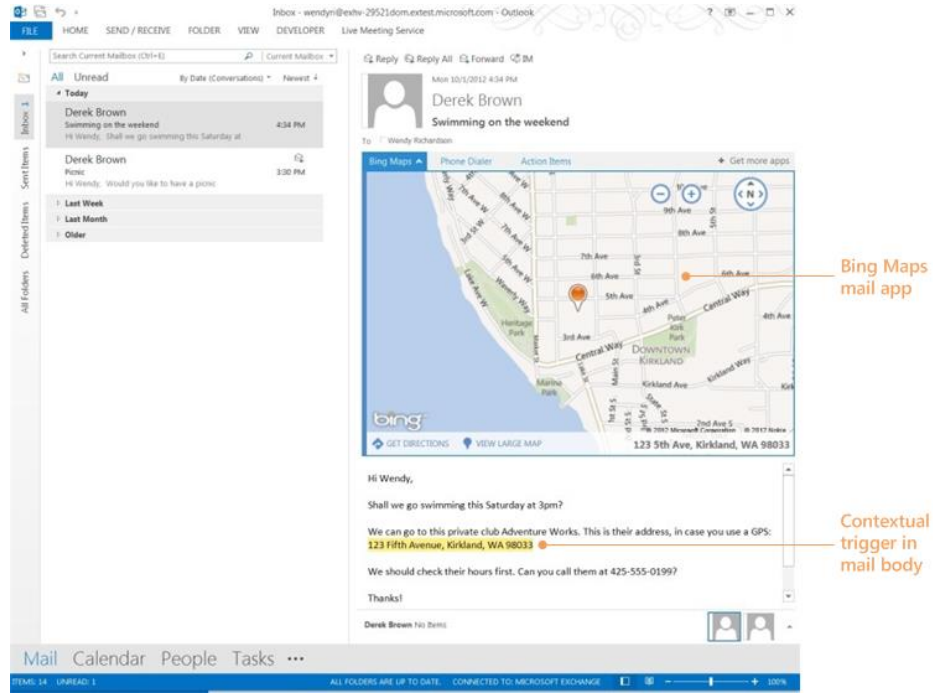
Task Pane Add-in



Content Add-in



Mail Add-in



Typescript vs js

Javascript:

PRO: familiar, popular, web language
widely supported in browsers

CON:
ES5, ES6, browser support, shims
lack of strong typing, late syntax/error

Typescript:

PRO: strongly typed, catches errors up front

Ready?

ScriptLab Workshop Goals:

Learn to use ScriptLab

Learn about Office JS API and add-ins (Excel)

Write code

Test

Share your code

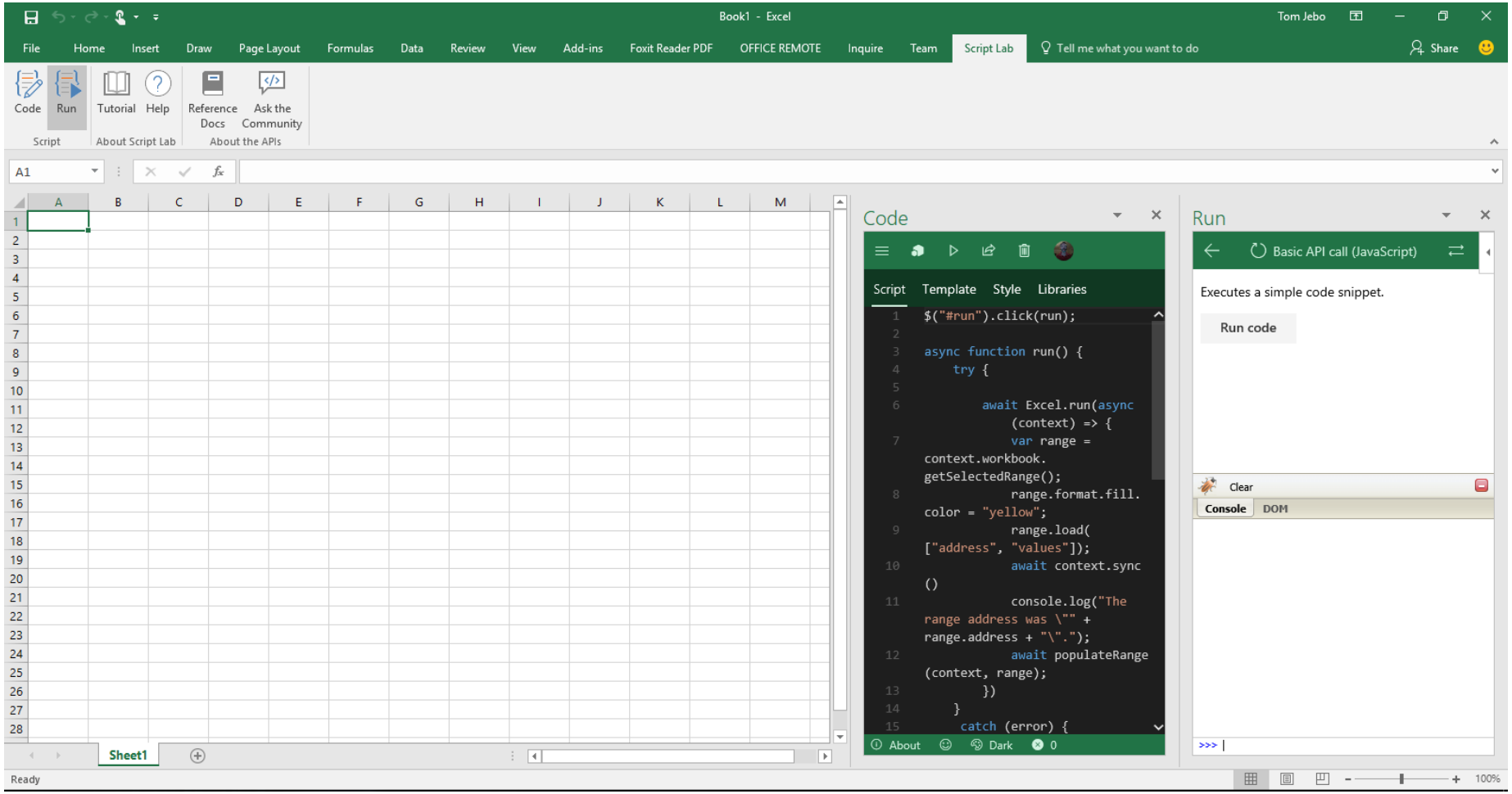
Get Set…

Getting setup:

1. Load Excel
   Preferrably Excel desktop but you can use online
2. Install the ScriptLab add-in from Store
3. Load the Office JS docs on
   http://dev.office.com/
   http://dev.office.com/devprogram - free 1 year subscription to Office
   https://dev.office.com/docs/add-ins/excel/excel-add-ins-javascript-programming-overview

4. Get the lab modules:
   https://github.com/tomjebo/addin-workshop

ScriptLab


Book1 - Excel
Tom Jebo
File Home Insert Draw Page Layout Formulas Data Review View Add-ins Foxit Reader PDF OFFICE REMOTE Inquire Team Script Lab
Tell me what you want to do
Share
Code Run Tutorial Help Reference Docs Ask the Community
Script About Script Lab About the APIs
A1
Sheet1
Ready
Code
Script Template Style Libraries
$("#run").click(run);
async function run() {
try {
await Excel.run(async (context) => {
var range = context.workbook.getSelectedRange();
range.format.fill.color = "yellow";
range.load(["address", "values"]);
await context.sync()
console.log("The range address was \"" + range.address + "\".");
await populateRange(context, range);
})
}
catch (error) {
About Dark 0
Run
Basic API call (JavaScript)
Executes a simple code snippet.
Run code
Clear
Console DOM
>>>
100%

context

```
var ctx = new Excel.RequestContext();
```

Requests to the Excel application

proxy objects

```
var selectedRange = ctx.workbook.getSelectedRange();
```

Trust them, they know what they're doing!

sync()

Synchronize between proxies and Excel.

Batched operations queued up get synced.

Excel.run()

Excel.run() executes a batch script that performs actions on the Excel object model.

This is where your code runs!

load()

```
object.load(string: properties);
//or
object.load(array: properties);
//or
object.load({loadOption});
```

You can chain together these calls to get a proxy and load values.

```
var range =
ctx.workbook.worksheets.getActiveWorksheet().getRange("A1:A2").load("values");
```

Async and promises

```
Excel.run(function (ctx) {

    return ctx.sync().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));
        }
});
```

Context object returns a promise. JS provices the .then construct.

Go!

Module 1	ScriptLab
	Run Basic API call (js)
	Populate cells
	MySnippets

Async and Typescript

```
async function run() {
try {
        await Excel.run(async (context) => {
            // some code ...
            await context.sync()

            console.log("The range address was \"" + range.address + "\".");
            await populateRange(context, range);
        })
    }
catch (error) {
        OfficeHelpers.UI.notify(error);
        OfficeHelpers.Utilities.log(error);
    }
}
```

Use async and await construct.
More readable!

Better, faster, smarter!

Module 2	Typescript exercise
	Run Copy and Multiply Values sample
	Grand Total
	Tax

Charts

Module 3

Add a chart

Hints:

- Use the chart collection add method.
- See https://dev.office.com/reference/add-ins/excel/chartcollection

Functions

Module 4

Range.Calculate() and calculate():

See： https://dev.office.com/reference/add-ins/excel/range

ConditionalFormat object:

See: https://github.com/OfficeDev/office-js-docs/blob/ExcelJs_OpenSpec/reference/excel/conditionalformatcollection.md

Calling Graph

Module 5

- Create a new add-in
- Register the add-in application for use with Graph permission scopes
- Use the Office JS Helpers to authenticate
- Call Graph to send email

https://dev.office.com/docs/add-ins/excel/excel-add-ins-javascript-api-reference?product=excel

https://dev.office.com/docs/add-ins/excel/excel-add-ins-javascript-programming-overview?product=excel#excelrunfunctioncontext--batch-

Thank You!

Questions?