# Title: Outlook Add-on for freshsales

## Basic Overview

We have developed a dedicated addin for freshsales in the **outlook** marketplace. This addin currently enables agents to perform the following functionality

* For customers who don't want to connect their mailbox with Freshsales but still want to create new entities within their Freshsales account or users using multiple mailboxes for emailing their customer.
* Creation of new entity Lead/Contact.
* View information of recipients of the mail thread to bring more context into the communication.
* View related activities of the entity within the plugin.
* View related deals and their information for contacts.
* Add and view notes for an existing entity related to the mail thread or for future reference.
* Add Sales activities against the Lead/ Contact.
* Add tags against the entity.
* Ability to open the landing page of the entity in CRM.

## Tech Arch Overview

We have built this independent of our Freshsales product codebase.  This [branch](https://github.com/freshdesk/freshsales-outlook-addon) holds the code base for this addin.

**Tech Stack & Components**

* Reactjs(recommended by Microsoft)
* Office manifest frameworks
* Webpack for build management
* appsforoffice.microsoft.com/lib/1/hosted/Office.js
* https://cdn.jsdelivr.net/npm/moment@2.24.0/moment.min.js
* <https://developer.microsoft.com/en-us/fabric#/controls/web> (recommended by Microsoft. [link](https://developer.microsoft.com/en-us/fabric/#/components))
* AWS js SDK(easy upload for compiled files)(TBD).

## Outlook Frontend API reference

This addin is built for **Outlook** email inbox client and consumes Outlook API and hooks as mentioned [here](https://docs.microsoft.com/en-us/outlook/add-ins/apis?context=office/dev/add-ins/context). The important catch here is the versioning of the API requirements set. The current supported APIs and their docs are [1.6 docs](https://docs.microsoft.com/en-us/office/dev/add-ins/reference/objectmodel/requirement-set-1.6/outlook-requirement-set-1.6) and [1.4 docs](https://docs.microsoft.com/en-us/office/dev/add-ins/reference/objectmodel/requirement-set-1.4/outlook-requirement-set-1.4).  
  
Notes of the APIs that are widely being used in our Addin:-

1. [notificationMessages.replaceAsync](http://Office.context.mailbox.item.notificationMessages.replaceAsync): Used to replace inline **notification messages** at Compose level in the Outlook app, if any were there previously, else create new ones.
2. [roamingSettings.set()](https://docs.microsoft.com/en-us/javascript/api/outlook/office.RoamingSettings?view=outlook-js-preview#set_name__value_) - Used to set **key, value** pairs to Outlook context under the app signature. Doing this will only set the values locally but not at the Outlook level. We need to call the below API to persist at the Outlook level.
3. [roamingSettings.saveAsync()](https://docs.microsoft.com/en-us/javascript/api/outlook/office.RoamingSettings?view=outlook-js-preview#saveAsync_callback_) - This needs to be called after set to persist the values that are modified using ***set()***.  
   Note:- roamingSettings in outlook's words - Used for saving Outlook add-in-specific custom settings to the user's mailbox where the add-in is installed.

## Outlook Manifest XML Elements

Few Critical Elements that are crucial to the manifest are:-

1. [VersionOverrides](https://docs.microsoft.com/en-us/office/dev/add-ins/reference/manifest/versionoverrides#:~:text=The%20root%20element%20that%20contains%20information%20for%20the,defined%20in%20the%20VersionOverrides%20v1.0%20or%20v1.1%20schema.)  
   This is the place where we define our Ribbon commands (Not to be confused with Ribbon API), Like opening our addin from compose/edit modal, quick add Bcc, etc. There can also be multiple versions of this Element, to provide backward compatibility for older versions of the outlook.
2. [Requirement Sets](https://docs.microsoft.com/en-us/office/dev/add-ins/develop/office-versions-and-requirement-sets)  
   This is a subset of VersionOverrides and It allows us to define the APIs version which we intend to support. And in the underlying Action Elements, we can define our commands based on this version.   
   [API reference](https://docs.microsoft.com/en-us/office/dev/add-ins/reference/requirement-sets/outlook-api-requirement-sets)

An Example of the Manifest, with Two VersionsOverrides Elements and Requirement Sets, [reference link](https://docs.microsoft.com/en-us/office/dev/add-ins/reference/requirement-sets/outlook-api-requirement-sets):-



## Local App Setup

  1. Follow the [readme](https://github.com/freshdesk/freshsales-outlook-addon/blob/master/README.md) in the repo to locally up the app.

  3. Since outlook whitelists HTTPS requests, we need **ngrok** proxy for the locally running app with the custom domain. update manifest.xml with a custom domain at various touchpoints.

  2. upload the manifest.xml into custom addin options (web and desktop client goes through a different process to install it, but the manifest for both is the same)

## Windows Installation on Mac

In order to test the add-in outlook windows app, we may require windows installed in our mac machines.

* Please follow the steps in this video to install windows in mac - [Installing windows on mac](https://youtu.be/Hmm9Q-T0oTo)
* After installing windows on mac, please install office from below setup file

[OfficeSetup.exe](/display/freshsales/Outlook+Add-on+for+freshsales?preview=%2F230071734%2F294466395%2FOfficeSetup.exe)

* Open outlook from the office app and install the freshsales add-in

## Unit Testing / E2E Testing

We have implemented Unit-testing using [Jest](https://jestjs.io/), It is recommended by React team, to test the UI. Along with Jest, we have [*@testing-library/react*](https://testing-library.com/) for testing user interactions with the UI. [*Puppeteer*](https://pptr.dev/) *,*Which is an E2E framework And [snapshot-diff](https://github.com/jest-community/snapshot-diff), which compares your dom before and after an interaction has happened.

* Kindly follow the steps to get started with unit-testing, [Unit-testing](https://github.com/freshdesk/freshsales-outlook-addon/blob/master/README.md#unit-tests).
* Kindly follow the steps to get started with E2E test for FE, [E2E Frontend](https://github.com/freshdesk/freshsales-outlook-addon/blob/master/README.md#run-automation-test-suite).

Few useful concepts :-

* With regard to unit-testing, As unit-tests do not spawn a browser instance and render the component visually. The only way to test the UI is through CLI . And proceed with testing the DOM (As a dev is by printing). Having said that,  
    
  1. To actually print the dom on the CLI, Read this - <https://testing-library.com/docs/queries/about/#debugging>.  
    
  2. To query and test particular elements of the DOM, use this concept - <https://testing-library.com/docs/queries/bytestid>.   
    
  3. The examples here <https://react-testing-examples.com/> (I found them a bit outdated, like using deprecated APIs but not a Blocker)and here <https://testing-library.com/docs/react-testing-library/example-intro>, Could help one to understand how to go about writing BDD test cases.  
    
  4. To know more about Snapshot diff testing, <https://www.robinwieruch.de/jest-snapshot-test-difference>.
* Coming to Puppeteer,  
    
  1. We are using [Puppeteer-core](https://pptr.dev/#?product=Puppeteer&version=v8.0.0&show=api-puppeteer-vs-puppeteer-core), which uses existing installation of Chromium build.  
    
  2. To pre-record the user interactions in the browser, We are using [chrome extension for Puppeteer](https://www.checklyhq.com/docs/headless-recorder/) to generate and give us a JS macro script, Which we then modify to our testing needs. For Example, we need to add Input Actions( this is not recorded in the macro), based on the macro. So this is kind of mix and win situation. Where the developer has to understand what works and what does not, And refactor the macro.