Office Mail App Compose Demo

# Pre-Demo Setup

* + Optimal resolution: 1920 x 1080
  + Standard resolution: 1366 x 768

# Demo Snippets

For the purposes of the demo, I add these to the VS Toolbox, with heading as the snippet name

## Mail-HTML

<div id="content-main">

<div class="padding">

<h1>Warranty Info</h1>

<div>

<label>Products found for customer</label>

<div id="customerName" class="values" />

</div>

<div id="products">

<!--

<table id="products">

<tr>

<th>Product</th>

<th>Warranty End Date</th>

</tr>

<tr class="products">

<td>Watch</td>

<td>5/30/15</td>

</tr>

<tr class="products">

<td>Lawn Mower</td>

<td>8/10/18</td>

</tr>

<tr class="products">

<td>F20 Mountain Bike</td>

<td>7/1/13</td>

</tr>

</table>

-->

</div>

<div class="tiptext">Click the row to insert the warranty response in the users email</div>

</div>

</div>

## Mail-Call CRM

#region Call our CRM system

retVal.FirstName = "Steve";

retVal.LastName = "Lasker";

retVal.CustomerEmail = "SteveLas@microsoft.com";

TimeSpan purchaseDate = new TimeSpan(700, 0, 0, 0);

retVal.Products.Add(

new Models.ProductWarrantyStatus() {

Name = "Aeroflite Three-Hand Watch",

PurchasedDate = DateTime.Now.Subtract(purchaseDate),

WarrantyEndDate = DateTime.Now.Subtract(purchaseDate).AddYears(3)

});

retVal.Products.Add(

new Models.ProductWarrantyStatus() {

Name = "Topaz Necklace",

PurchasedDate = DateTime.Now.Subtract(purchaseDate).AddDays(200),

WarrantyEndDate = DateTime.Now.Subtract(purchaseDate).AddDays(200).AddYears(1)

});

retVal.Products.Add(

new Models.ProductWarrantyStatus() {

Name = "Tesla Model S",

PurchasedDate = DateTime.Now.Subtract(purchaseDate).AddDays(500),

WarrantyEndDate = DateTime.Now.Subtract(purchaseDate).AddDays(500).AddYears(5)

});

#endregion

## Mail-getCustomer()

function getCustomer() {

Office.cast.item.toItemCompose(Office.context.mailbox.item).to.getAsync(function (result) {

var customerEmail = result.value[0].emailAddress;

// Get the customer info from the CustomerInfo Service

var queryURL = '../../api/CustomerInfo/?customerEmail=' + customerEmail;

$.getJSON(queryURL, function (customerInfo) {

// Set the Customer Name

$('#customerName')[0].innerHTML = customerInfo.FirstName + " " + customerInfo.LastName;

getProducts(customerInfo.Products)

});

});

}

## Mail-GetProducts()

// Iterate over the products to produce a list to select from

// The list will contain the product name and it's warranty status

// which determines which email to send the customer

function getProducts(products) {

// Build a product HTML Table to display the list

var $table = $('<table class="productWarrantyStatus" />');

// Add the header

var $headerRow = $('<tr />').appendTo($table);

$('<th />').text("Product").appendTo($headerRow);

$('<th />').text("Warranty End Date").appendTo($headerRow);

// Add a row for each product, with formatting and a click event handler

$.each(products, function (index, item) {

// Add the row and set the style

var $row = $('<tr />').appendTo($table);

$row.addClass('products');

// add the click handler for all the work to repair the item

$row.click(function (e) {

// use the class name to easily determine if the product is in warranty,

// and insert the appropriate email text

var inWarranty = e.currentTarget.children[1].className == "inWarranty";

insertWarrantyInfo(inWarranty);

});

// Creat the Product and warranty end date columns

var $productCol = $('<td />').appendTo($row);

var $warrantyEndCol = $('<td />').appendTo($row);

// set the text of the two columns

$productCol.text(products[index].Name);

// set the style based on whether the product is under warranty

var date = moment(products[index].WarrantyEndDate);

$warrantyEndCol.text(date.format("MMM Do, YYYY"));

if (date.isBefore(new moment())) {

$warrantyEndCol.addClass('outOfWarranty')

} else {

$warrantyEndCol.addClass('inWarranty')

}

});

// Add the products table to the products Div tag

$table.appendTo($('#products'));

}

## Mail-insertWarrantyInfo()

function insertWarrantyInfo(inWarranty) {

var item = Office.context.mailbox.item;

var responseType;

if (inWarranty) {

responseType = "inWarranty";

} else {

responseType = "notUnderWarranty";

}

var queryURL = '../../api/CustomerResponse/' + responseType;

$.get(queryURL, null, function (responseText) {

item.body.setSelectedDataAsync(responseText,

{

coercionType: Office.CoercionType.Html,

asyncContext: { var3: 1, var4: 2 }

});

});

}

## Mail-Styles

/\* Common app styling \*/

#content-header {

background: #2a8dd4;

color: #fff;

position: absolute;

top: 0;

left: 0;

width: 100%;

height: 80px; /\* Fixed header height \*/

overflow: hidden; /\* Disable scrollbars for header \*/

}

#content-main {

background: #fff;

position: fixed;

top: 80px; /\* Same value as #content-header's height \*/

left: 0;

right: 0;

bottom: 0;

overflow: auto; /\* Enable scrollbars within main content section \*/

}

.padding {

padding: 15px;

}

#notification-message {

background-color: #818285;

color: #fff;

position: absolute;

width: 100%;

min-height: 80px;

right: 0;

z-index: 100;

bottom: 0;

display: none; /\* Hidden until invoked \*/

}

#notification-message #notification-message-header {

font-size: medium;

margin-bottom: 10px;

}

#notification-message #notification-message-close {

background-image: url("../Images/Close.png");

background-repeat: no-repeat;

width: 24px;

height: 24px;

position: absolute;

right: 5px;

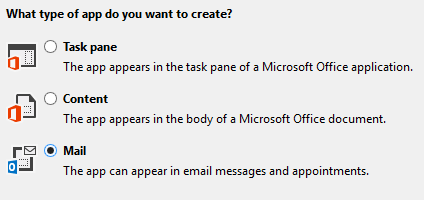
top: 5px;

cursor: pointer;

}

Mail Content App

# Create New Project

* Name: **MailComposeApp**
* Office Apps  
  
* Focus on Email
* Allow Read/Compose for Email/Calendar
* Finish

# F5 – Default

## Talking Points

* Side load the mail app
* Can run in the browser or desktop client
* Although the app contents is HTML, it looks like it just fits
* The Office style sheet provides the styles you need
* We can read from/write to the mail or calendar item

# Review VS Project

## Talking Points

* Same Model of a Manifest for the store, and website for the app
* Web Project is just a Web Project, leveraging HTML5/JavaScript
* Includes Office.js libraries
* jQuery libraries
* Review **Manifest Properties**
* Re **Project properties**

# Update the App

Lets build our app

## Talking Points

* We’re going to build an app that helps customer service reps respond to repair requests
* Image we receive email from customers. They want their item repaired, but don’t know if it’s still under warranty
* Think of the steps the customer service person must do
  + Look up the customer, by name/email
  + Find the products they’ve purchased
  + Calculate if the item is still under warranty
  + Provide them information for how to return the item
  + Send them a package to send the item back
  + Create the entries in their CRM system
* Can we automate this?

# Lookup the customer

* Using the email, lets lookup the customer
* Once we find the customer, list the products they’ve purchased
* Visually indicate if the item is under warranty
* When the rep clicks the item, lets paste in the response and do the automation in our CRM system

# Update the HTML

* Paste in the HTML

# Add the CustomerInfo Service

## Talking Points

* Lets use WebAPI for our client to call back to the server to find the customer
* Because this is an ASP.net web server project, we can add all the normal service stuff

## Start the GetCustomer

* Open the Home.js file – our code behind
* Review the file
* Remove the boiler plate code
* Add the getCustomer call

function getCustomer() {}

* Retrieve the email of the user to find them in our CRM system
* This is where we’re using the Office APIs

Office.cast.item.toItemCompose(Office.context.mailbox.item).to.getAsync(function (result) {});

* Using the item, pull the first email

var customerEmail = result.value[0].emailAddress;

* Where are we going to get the customer from?

## Add ServiceController

* Create a **folder** called **Controllers**
* Create a **folder** named **Models**
* Add the Models for our CustomerInfo
  + Drag contents of Models to project
* Notice these are just normal POCO objects we’ll return
* Create the Get method on our Service Controller

public Models.CustomerWarrantyInfo Get(string customerEmail) {}

* Instance the CustomerWarrantyInfo and return it

Models.CustomerWarrantyInfo retval = new Models.CustomerWarrantyInfo();

return retval;

* Drag the contents of our CRM call
* The details aren’t as important as we’re in ASP.net. Nothing special here, so lets focus on the Office Mail App

# Consume the service

* Remember, there can be multiple emails in the to

// Get the customer info from the CustomerInfo Service

var queryURL = '../../api/CustomerInfo/?customerEmail=' + customerEmail;

* Notice we’re using a relative path, and our **api** routing – again, standard WebAPI code
* We pass in the customer email, with the named parameter (optional)
* Add a jQuery getJSON to call our WebAPI service, passing in the URL and setting up a parameter for the result

$.getJSON(queryURL, function (customerInfo) {});

* Within the async function, lets first set the customer name in our HTML

// Set the Customer Name

$('#customerName')[0].innerHTML = customerInfo.FirstName + " " + customerInfo.LastName;

* What’s special about this? – Nothing, it’s jQuery, HTML/JavaScript
* Lets now format the products that get returned
* Drag the getProducts clip from the toolbox

## Call GetCustomer on App Load

* In initialize method, call getCustomer

## Style Sheet & moment.js

* Add moment.js for date manipulation
* Add some styles to the hom.css

# F5

* Set a breakpoint in the app startup

# Add the InsertWarrantyInfo

* Drag in the CustomerResponseController
* Drag in the GetWarrantyResponse snippet

# F5