|  |
| --- |
| Microsoft SharePoint 2013 - Hands-on Lab |
| Mail Apps |
| Verified Against Build 15.0.4420.1017 |

|  |
| --- |
| Microsoft  Version 1.0  August 14, 2012 |

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links are provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Copyright © 2012 Microsoft Corporation. All rights reserved.

Microsoft and SharePoint are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The names of actual companies and products mentioned herein may be the trademarks of their respective owners.

# Technical Background

Mail Apps enable you to create apps that are deployed to Exchange on-premises or in the cloud, and the apps appear in Outlook either in the desktop client or in the browser. Mail Apps are triggered based on rules that the developer leverages in the app.

## Create Office 365 Developer SharePoint Site Collection

This lab requires you first create an Office 365 Developer Site, which provisions Exchange as part of your subscription. Visit <http://dev.office.com> and sign up for an Office 365 Developer Site. Once provisioned, create a new site using the Developer Site template at [**https://YourTenant.sharepoint.com/sites/Dev**](https://YourTenant.sharepoint.com/sites/Dev)**.**

# Introduction

## Estimated time to complete this lab

30 minutes

## Objectives

After completing this lab, you will be able to:

* Create a Mail App and deploy it to Office 365.
* View the Mail App in the desktop client.
* View the Mail App in the browser.

## Overview of Lab

Mail Apps enable you to create apps that are deployed to Exchange on-premises or in the cloud, and the apps appear in Outlook either in the desktop client or in the browser. Mail Apps are triggered based on rules that the developer leverages in the app. In this lab, we will create an app that is triggered based on conditions of an incoming email message.

## Create Office 365 Developer SharePoint Site Collection

This lab requires you first create an Office 365 Developer Site. Visit <http://dev.office.com> and sign up for an Office 365 Developer Site. Once provisioned, create a new site using the Developer Site template at [**https://YourTenant.sharepoint.com/sites/Dev**](https://YourTenant.sharepoint.com/sites/Dev)**.**

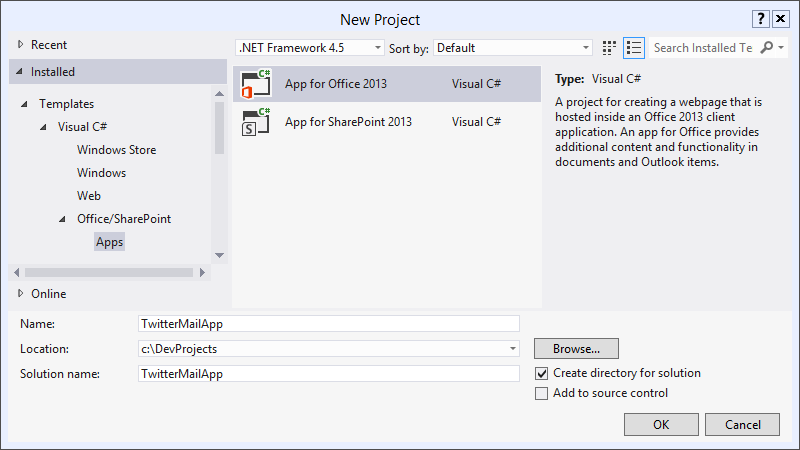
# **Exercise 1: Hello Mail App**

In this exercise, you will create a simple Twitter Mail App that will look for Twitter screen names in an email body and will display the recent tweets for that screen name.

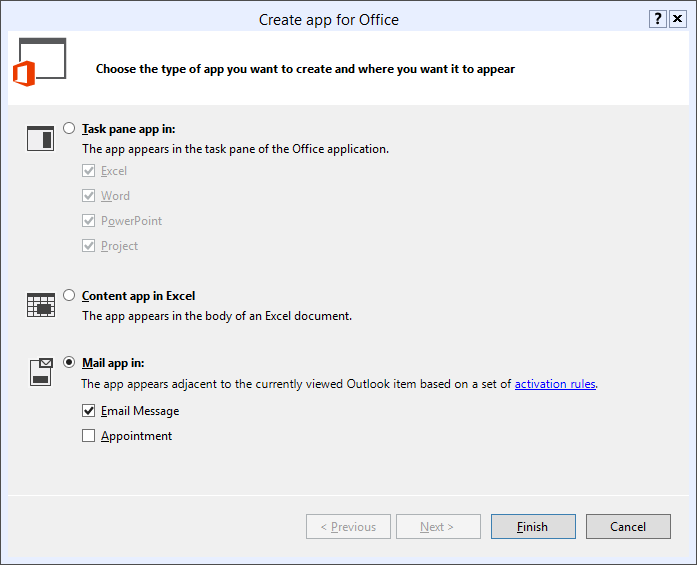
## Task 1 – Create a New App

In this task, you will develop a new App in Visual Studio

1. Open Microsoft Visual Studio 2012 and create a new SharePoint App project
   1. Open **Microsoft Visual Studio 2012**
   2. Select **File⮚New Project** from the main menu
   3. Click the **Templates⮚Visual C#⮚Office/SharePoint⮚Apps** node and select the **App for Office 2013** project template
   4. Name the new project **TwitterMailApp.**



* 1. Click the **OK** button.
  2. In the next screen, choose **Mail app in** and choose **Email Message.**



## Task 2 – Implement the App Web

In this task, you will modify the generated ASP.NET web application. You will modify the HTML page structure and provide the javascript implementation for the app.

1. In the **TwitterMailAppWeb** project, open the **TwitterMailApp.html** page. Leave the content in the **head** section as-is. Modify the content in the **body** section:

<div>

<p id="message">

<!-- The following content will be replaced with the user name when you run the app -->

initializing...

</p>

</div>

1. Modify the **TwitterMailApp.js** file with the following:

// This function is run when the app is ready to start interacting with the host application

// It ensures the DOM is ready before updating the span elements with values from the current message

Office.initialize = function () {

$(document).ready(function () {

init(Office.context.mailbox.item.getRegExMatches().ScreenName);

});

};

function init(screenNames) {

for (var i = 0; i < screenNames.length; i++) {

getTwitterData(screenNames[i]);

}

}

function getTwitterData(screenName) {

$.ajax({

url: "https://api.twitter.com/1/statuses/user\_timeline.json",

type: "GET",

dataType: "jsonp",

data: {

screen\_name: screenName,

include\_rts: true,

count: 5,

include\_entities: true

},

success: onDataReturned,

error: onError

});

}

function onDataReturned(data) {

var ret = [];

ret.push("<h1>Tweets for " + data[0].user.screen\_name + "</h1>");

ret.push("<table><th>Text</th><th>Time</th><tr>");

for (var i = 0; i < data.length; i++) {

ret.push("<tr>");

ret.push("<td>" + data[i].text + "</td>");

ret.push("<td>" + data[i].created\_at + "</td>");

ret.push("</tr>");

}

ret.push("</table>");

$('#message').html(ret.join(''));

}

function onError(data, something, another) {

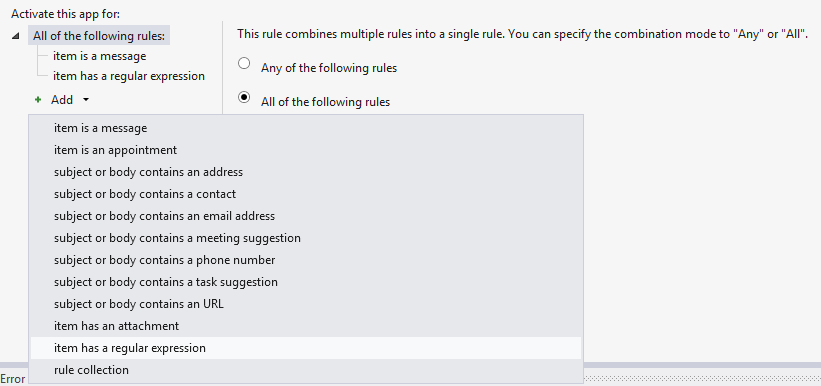
alert(data);

}

## Task 3 – Modify the app manifest

In this task, you will modify the app manifest to provide the location of the app and the activation rules for the app.

1. Open the **AppManifest** designer by double-clicking on the **TwitterMailApp** node in solution explorer.
2. Open the **Activation Rules** tab to set activation rules for the app.
   1. Click the **Any of the following rules** node and change the radio button to select **All of the following rules.**
   2. Click the **Add** node to add a new activation rule. In the drop-down, choose **item has a regular expression**.



1. Click the **item has a regular expression** node, and change the property name to **BodyAsPlaintext**.
2. Change the RegEx name to **ScreenNames**. Use the simple regular expression:

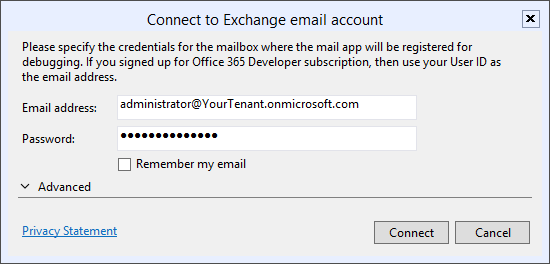
**SPConf|SharePoint|andrewconnell|kaevans**

1. Check the checkbox to ignore case.

## Task 4 – Test the App

In this task, you will run and test the App. We deploy the app manifest to Exchange, but the implementation of the app runs in our local IIS Express development server.

* 1. Select **Debug⮚Start Debugging** in Visual Studio.
  2. When prompted for credentials, enter your Office 365 User ID as the email address and provide your password.



* 1. Observe the **Output Window** for installation messages, and verify that **Internet Explorer** opens to the SharePoint Apps site you created before starting the lab.
  2. Send yourself an email message that contains the body text:

**Can’t wait for SharePoint Conference!**

* 1. Verify that the HTML table is rendered with recent tweets from the @SharePoint twitter user.

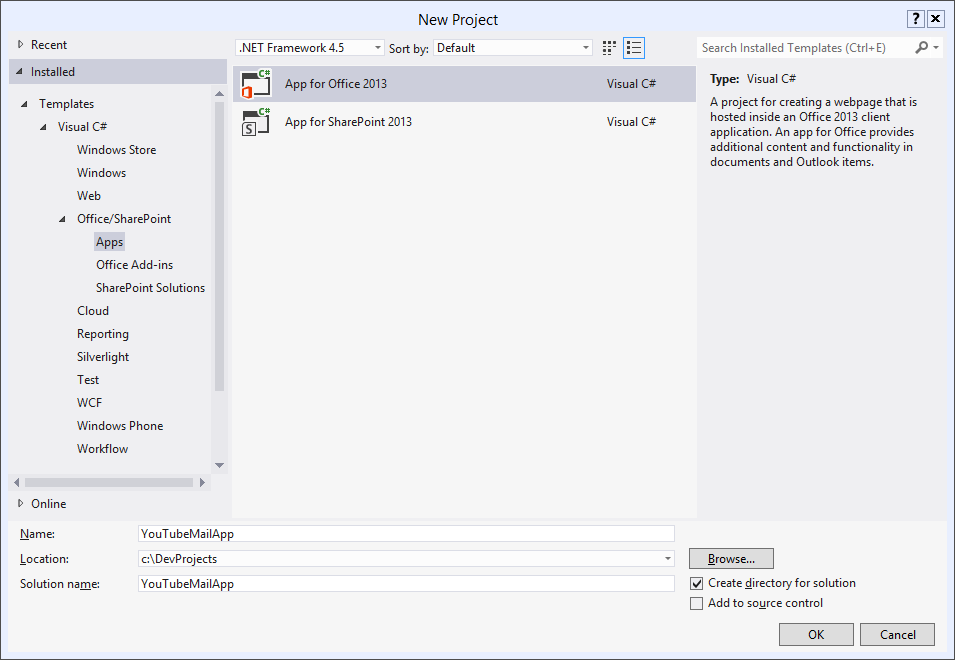
# **Exercise 2: YouTube Mail App**

In this exercise, you will create a Mail App that will allows users to view a YouTube video from a message or appointment item that contains the video URL, without leaving Outlook 2013.

## Task 1 – Create a New App

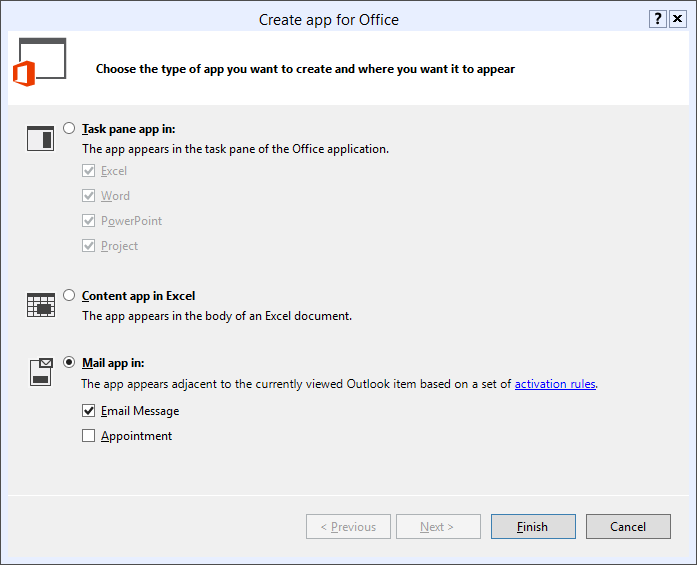
In this task, you will develop a new App in Visual Studio

1. Open Microsoft Visual Studio 2012 and create a new SharePoint App project
   1. Open **Microsoft Visual Studio 2012**
   2. Select **File⮚New Project** from the main menu
   3. Click the **Templates⮚Visual C#⮚Office/SharePoint⮚Apps** node and select the **App for Office 2013** project template
   4. Name the new project **YouTubeMailApp**
   5. Click the **OK** button



*Create new SharePoint App Project*

1. In the next screen, choose **Mail app in** as the type of app, and check the **Email Message** checkbox.

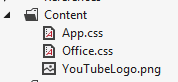


1. Click **Finish**.

## Task 2 – Implement the App Web

In this task, you will modify the generated ASP.NET web application. You will add a logo for the app, provide the CSS styles for the app, modify the HTML page structure, and provide the javascript implementation for the app.

1. Two projects are created, **YouTubeMailApp** and **YouTubeMailAppWeb**. Add an the YouTubeLogo.png image from the **Required Files** folder of the lab documents to the **Content** folder in YouTubeMailAppWeb.



1. **Edit** the file **App.css**. **Replace** the contents with the following:

\*

{

margin: 0px;

overflow: hidden;

}

.thumbnailFrame

{

margin-bottom: 4px;

}

.thumbnail

{

margin-right: 4px;

}

.videoTitle

{

font-family: Segoe UI;

font-size: 16px;

}

.multiLineVideoDetails

{

font-family: Segoe UI;

font-size: 14px;

color: Gray;

}

.singleLineVideoDetails

{

font-family: Segoe UI;

font-size: 12px;

color: Gray;

}

1. **Edit** the file **YouTubeMailApp.html**. Leave the **head** section as-is. **Replace** the **body** section with the following:

<table cellpadding="0px" cellspacing="0px" style="table-layout: fixed" height="200px">

<tr>

<td id="thumbnails" valign="top" />

<td id="embeddedVideo" valign="top" />

<td valign="top">

<div id="details" style="margin-left: 10px; height: 200px; overflow: auto" />

</td>

</tr>

</table>

1. Now it’s time to provide the implementation. **Replace** the contents of the **YouTubeMailApp.js** file with the following contents.

var videos;

var selectedVideo = -1;

Office.initialize = function (reason) {

$(document).ready(function () {

init(Office.context.mailbox.item.getRegExMatches().VideoURL);

});

}

function parseDate(dateString) {

var year = parseInt(dateString.substring(0, 4));

var month = parseInt(dateString.substring(5, 7));

var day = parseInt(dateString.substring(8, 10));

var result = new Date();

result.setUTCFullYear(year);

result.setUTCMonth(month - 1);

result.setUTCDate(day);

return result;

}

function getVideoIndex(videoId) {

for (i = 0; i < videos.length; i++) {

if (videos[i].Id == videoId) {

return i;

}

}

return null;

}

function videoDetailsLoaded(videoFeed) {

var videoIndex = getVideoIndex(videoFeed.entry.id.$t.substring(42));

if (videoFeed.entry.media$group.media$thumbnail.length > 0) {

videos[videoIndex].ThumbnailURL = videoFeed.entry.media$group.media$thumbnail[0].url.replace("http:", "https:");

document.getElementById(videos[videoIndex].Id).src = videos[videoIndex].ThumbnailURL;

}

videos[videoIndex].Title = videoFeed.entry.title.$t;

videos[videoIndex].PublishedDate = parseDate(videoFeed.entry.published.$t);

videos[videoIndex].Description = videoFeed.entry.media$group.media$description.$t;

videos[videoIndex].ViewCount = parseInt(videoFeed.entry.yt$statistics.viewCount);

if (videoIndex == selectedVideo) {

refreshVideoDetails(selectedVideo);

}

}

function loadVideoDetails(videoIndex) {

var script = document.createElement("script");

script.setAttribute("src", "https://gdata.youtube.com/feeds/api/videos/" + videos[videoIndex].Id + "?alt=json-in-script&callback=videoDetailsLoaded");

document.getElementsByTagName('head')[0].appendChild(script);

}

function refreshVideoDetails(videoIndex) {

var html = "";

if (videos[videoIndex].Title != undefined) {

html += "<p class='videoTitle'>" + videos[videoIndex].Title + "</p>";

}

if (videos[videoIndex].Description != undefined) {

html += "<p class='multiLineVideoDetails'>" + videos[videoIndex].Description + "</p>";

}

if (videos[videoIndex].PublishedDate != undefined) {

html += "<p class='singleLineVideoDetails' style='margin-top: 8px;'>" + videos[videoIndex].PublishedDate + "</p>";

}

if (videos[videoIndex].ViewCount != undefined) {

html += "<p class='singleLineVideoDetails'>" + videos[videoIndex].ViewCount + "</p>";

}

document.getElementById("details").innerHTML = html;

}

function selectVideo(videoIndex) {

selectedVideo = videoIndex;

for (i = 0; i < videos.length; i++) {

document.getElementById(videos[i].Id + "frame").style.background = i == videoIndex ? "Black" : "White";

}

// document.getElementById("embeddedVideo").innerHTML = "<iframe width='354' height='200' frameborder='0' src='https://www.youtube.com/embed/" + videos[videoIndex].Id + "?autohide=1&showinfo=0'/>";

document.getElementById("embeddedVideo").innerHTML = "<iframe width='354' height='200' frameborder='0' src='https://www.youtube.com/embed/" + videos[videoIndex].Id + "?html5=True'/>";

refreshVideoDetails(videoIndex);

}

function init(videoURLs) {

var html = "";

videos = new Array();

for (i = 0; i < Math.min(videoURLs.length, 5) ; i++) {

var questionMarkPosition = videoURLs[i].indexOf("?v=");

var videoId;

if (questionMarkPosition >= 0) {

videoId = videoURLs[i].substr(questionMarkPosition + 3, 11);

}

else {

videoId = videoURLs[i].substr(16, 11);

}

videos[i] = { "Id": videoId };

html += "<div class='thumbnailFrame' id='" + videos[i].Id + "frame'><img class='thumbnail' id='" + videos[i].Id + "' width='32' height='32' onclick='selectVideo(" + i + ");'/></div>";

loadVideoDetails(i);

}

document.getElementById("thumbnails").innerHTML = html;

if (videos.length == 1) {

document.getElementById("thumbnails").style.display = "none";

}

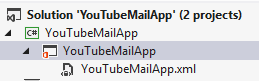
selectVideo(0);

}

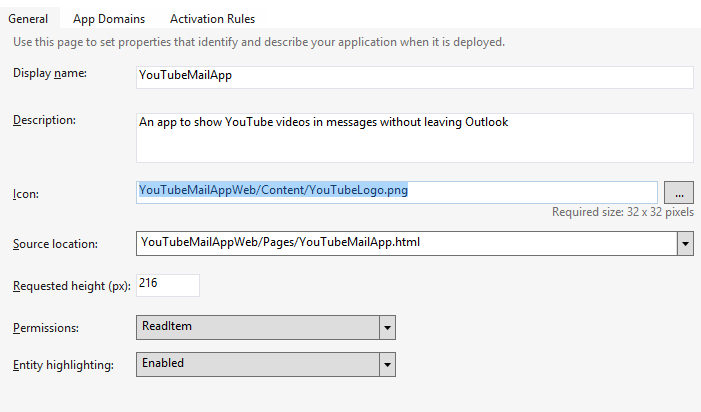
## Task 3 – Modify the App Manifest

In this task, you will modify the app manifest to point to the newly created app and will update the activation rules for the app.

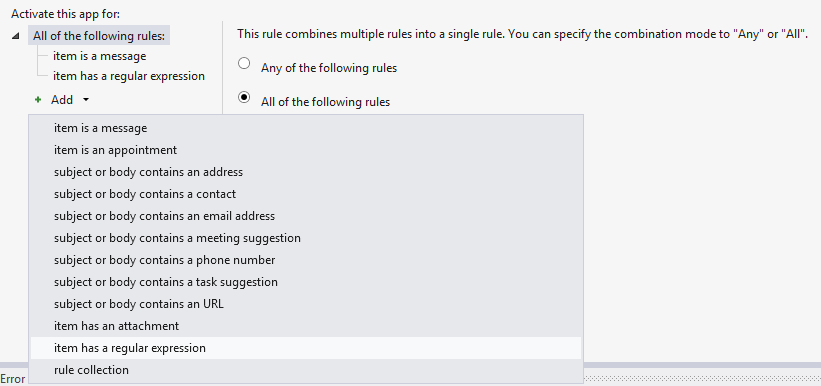
1. Open the **AppManifest** designer by double-clicking on the **YouTubeMailApp** node in the YouTubeMailApp project.



1. Provide an app icon and change the height.
   1. On the **General Settings** tab, change the **Icon** property to point to the **YouTubeLogo.png** file that was added to the app web.
   2. Use the value **YouTubeMailAppWeb/Content/YouTubeLogo.png** for the **Icon** property.
   3. Change the **Requested Height** to **216**.



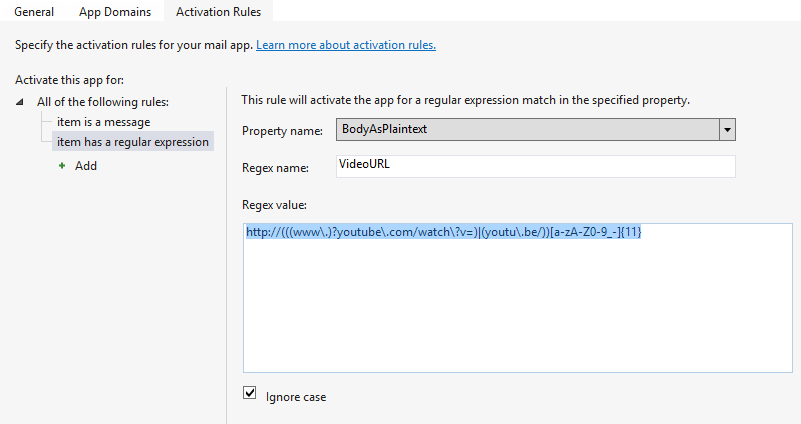
1. Open the **Activation Rules** tab to set activation rules for the app.
   1. Click the **Any of the following rules** node and change the radio button to select **All of the following rules.**
   2. Click the **Add** node to add a new activation rule. In the drop-down, choose **item has a regular expression**.



* 1. Click the **item has a regular expression** node, and change the property name to **BodyAsPlaintext**.
  2. Change the RegEx name to **VideoUrl**.
  3. Change the **Regex value** to the following:

**http://(((www\.)?youtube\.com/watch\?v=)|(youtu\.be/))[a-zA-Z0-9\_-]{11}**

* 1. Check the **Ignore case** checkbox.

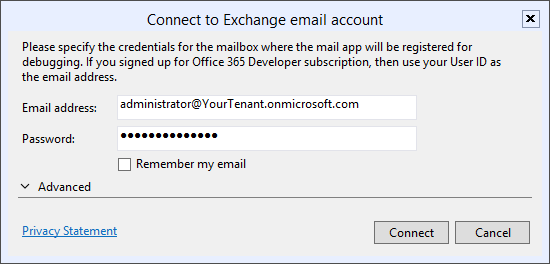


* 1. In the **File** menu in Visual Studio, choose **Save All** to save your work.

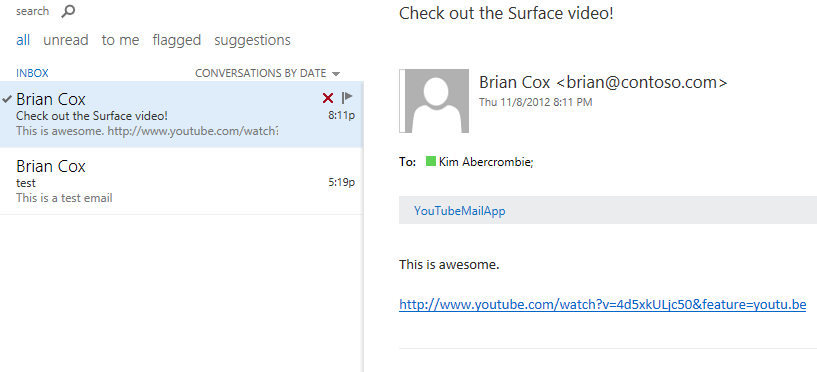
## Task 4 – Test the App

In this task, you will run and test the App. We deploy the app manifest to Exchange, but the implementation of the app runs in our local IIS Express development server.

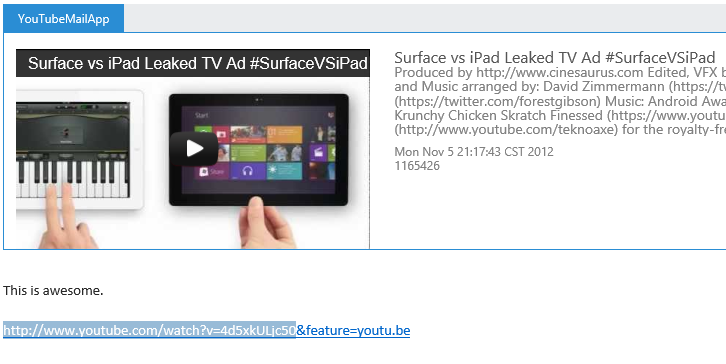
* 1. Select **Debug⮚Start Debugging** in Visual Studio.
  2. When prompted for credentials, enter your Office 365 User ID as the email address and provide your password.



* 1. Observe the **Output Window** for installation messages, and verify that **Internet Explorer** opens to the SharePoint Apps site you created before starting the lab.
  2. Send yourself an email message to your Office 365 inbox **without** a link to a YouTube video. Observe that the app does not appear in the email window.
  3. Send yourself an email message to your Office 365 inbox **with** a link to a YouTube video in the email body.



* 1. Click the link to the **YouTubeMailApp**. Notice the warning window that appears. This is because we are using our localhost development server that does not have a trusted certificate. Click the **Show content** button to continue.
  2. The app is rendered and we can see the thumbnail preview of the video along with details of the video, all within Outlook.



## Summary

In this lab, you learned how to create apps that are deployed to Exchange but the functionality executes on a separate server. You learned how to use regular expressions to activate an app and how to use the Office API to interact with an email message.