LightSwitch HTML Client Tutorial

Microsoft LightSwitch HTML Client for Visual Studio 2012

The new HTML5 and JavaScript-based client is an important companion to our Silverlight-based desktop client that addresses the increasing need to build touch-oriented business applications that run well on modern mobile devices.

In this tutorial, we’ll build a touch-first, modern experience for mobile devices. To help ground the tutorial, we’ve created a fictional company scenario that has a need for such an application.

Contents

[The Contoso Moving Application 2](#_Toc350155878)

[Implement the ContosoMoving App 6](#_Toc350155879)

[Step 1: Add a Companion HTML Client to an Existing Application 7](#_Toc350155880)

[Step 2: Define Core Screens and App Navigation 9](#_Toc350155881)

[Step 3: Create an Edit Dialog 17](#_Toc350155882)

[Step 4: Customize the UI 27](#_Toc350155883)

[Step 5: Test the Application on a Tablet Device 36](#_Toc350155884)

[Step 6: Using a Device’s Built-in Camera to Shoot and Upload Photos 53](#_Toc350155885)

[Step 7: Add a Bing Map Custom Control 60](#_Toc350155886)

[Step 8: Republish the App and Test it on a Tablet Device 63](#_Toc350155887)

[Step 9: Brand the Application 64](#_Toc350155888)

## Helpful resources

As you walk through this tutorial, please bear in mind that there are useful resources available to help you should you get stuck or have a question:

* [LightSwitch HTML Client forum](http://social.msdn.microsoft.com/Forums/en-US/lightswitchhtml/threads) to post feedback and ask questions, and check back for any corrections to the walkthrough document.
* [HTML Client Resources Page on the Developer Center](http://msdn.microsoft.com/en-us/lightswitch/htmlclient) for more learning resources.

# The Contoso Moving Application

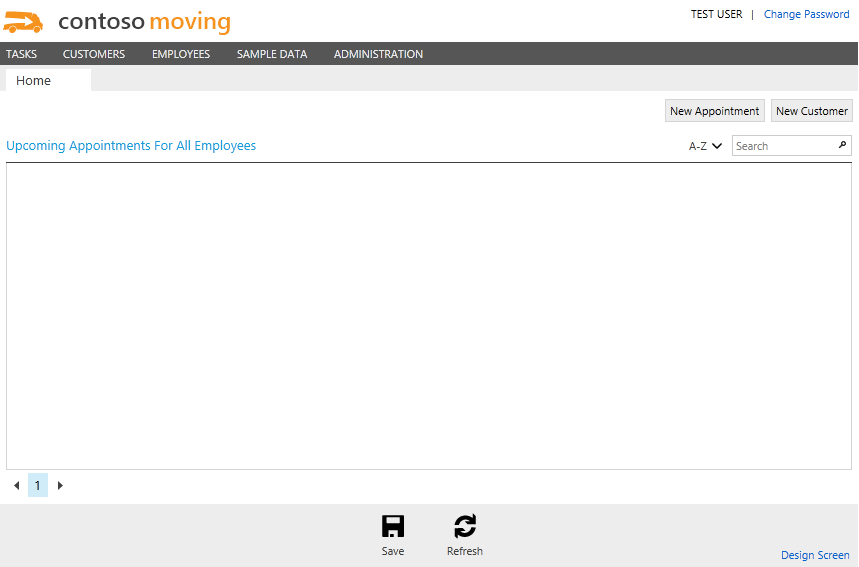
*Contoso Moving* is an application that’s used by *Contoso Movers, Inc.* to take the inventory of customers’ residences prior to moving. The data collected via the application helps *Contoso Movers* determine the resources required to move a particular client’s belongings—how many trucks, people, boxes, etc. need to be allocated. The application is comprised of two clients that serve distinct business functions:

1. **Schedulers use a desktop application** to service new customer requests and create appointments. This application is a rich desktop application primarily geared towards heavy data entry with the keyboard and mouse, since Schedulers are on the phone with customers a lot and need to enter quite a bit of information into the system during the course of a day.
2. **Planning Specialists use a tablet device** to quickly take inventory—on location—of each residence on the specialist’s schedule for the day. Taking inventory involves detailing each room in the residence, its size and entry requirements (if any), and listing its contents. Pictures are often taken of each room so the movers have a point of reference when they arrive. Secondarily, Planning Specialists may make notes about parking restrictions for the move team (i.e., where they can park the truck during the move).

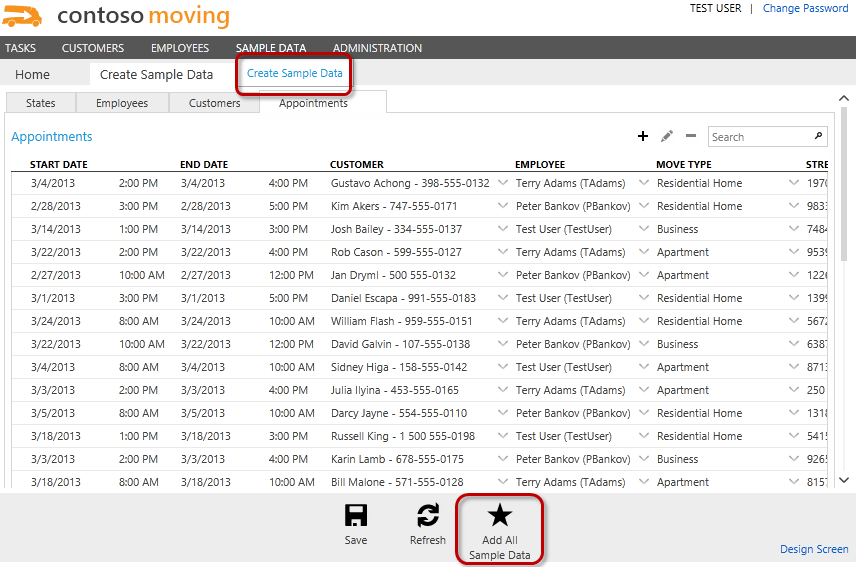
This tutorial walks through building out the mobile client used by *Contoso Movers’* planning specialists. Before we do that, let’s take a look at the rich desktop client provided by the sample.

## Run the Desktop Client for Schedulers

Open the solution provided by the sample in **Visual Studio 2012** and press **F5** to run the application. You will see the desktop client launched in a browser. Currently there is no data in the client.

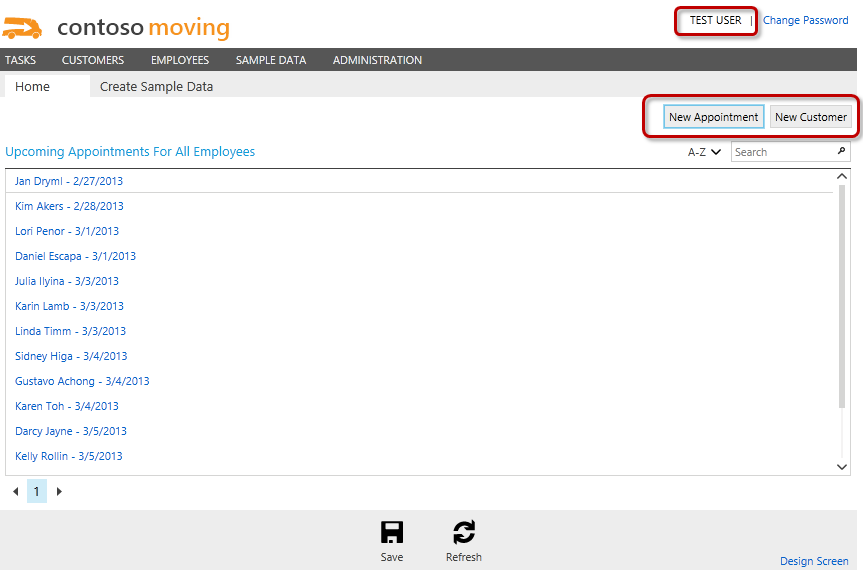


To speed things up, the sample provides a functionality to add some sample data for us to work with. From the navigation menu, select **SAMPLE DATA**, open **Create Sample Data** screen. Click **Add All Sample Data** command at the bottom. This will populate the local database with sample data for our walkthrough.



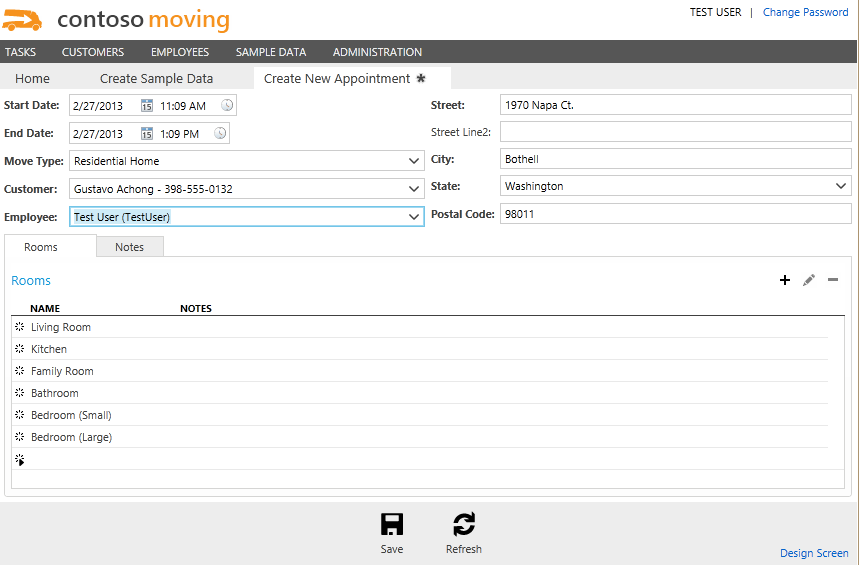
After data is added, go back to the home screen and hit the **Refresh** button. The upcoming appointments are now displayed in the home screen. Please note that *Contoso Moving* uses **Forms authentication**. Therefore, in debug mode, you (as a developer) are logged in as **TEST USER**. You can see the current user on the top right corner of the client.

Two key entry points—**New Customer** and **New Appointment**—are provided for entering the details of a new customer and setting up new appointments.



Choose the **New Appointment** button to create a new appointment. The **Create New Appointment** screen is used to capture information about the appointment such as preferred date/time, address, and contact information. Typically, a customer will provide enough information over the phone to create a rough profile about the move. For example, is the move for a residential home, apartment, or office building, the rooms involved in the move, and any special instructions or questions that the Planning Specialist should follow up with the customer about.

Feel free to add several of your own appointments in the system, as well as explore other screens accessible from the menu bar.



# Implement the ContosoMoving App

Now we’re ready to build a companion HTML client to the application. Topics we’ll cover throughout the rest of the tutorial are broken into 2 main parts:

**Part 1: The Basics**

[Step 1: Add a Companion HTML Client to an Existing Application](#_Step_1:_Add)  
[Step 2: Define Core Screens and App Navigation](#_Step_2:_Define)  
[Step 3: Create an Edit Dialog](#_Step_3:_Create)  
[Step 4: Customize the UI](#_Step_4:_Customize)  
[Step 5: Test the Application on a Tablet Device](#_Step_5:_Test)

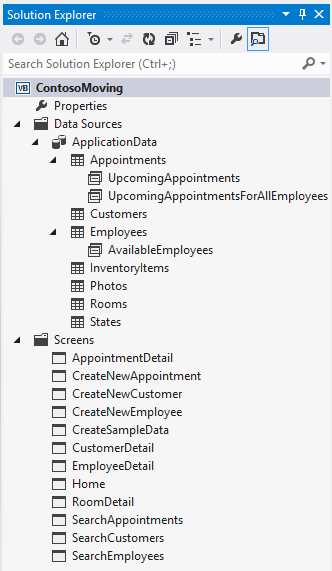
**Part 2: More Customizations**

[Step 6: Using a Device’s Built-in Camera to Shoot and Upload Photos](#_Step_6:_Using)   
[Step 7: Add a Bing Map Custom Control](#_Step_7:_Add)  
[Step 8: Republish the App and Test it on a Tablet Device](#_Step_8:_Republish)  
[Step 9: Brand the Application](#_Step_9:_Brand)

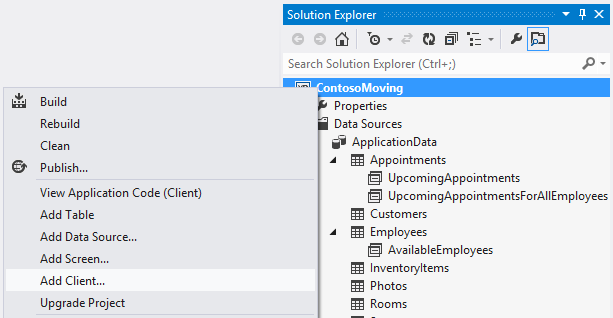
Let’s get started!

# Step 1: Add a Companion HTML Client to an Existing Application

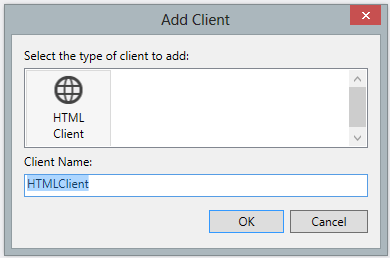
1. If we look at **Solution Explorer,** we see that the project already contains the core data service logic of the application (located under the **Data Sources** node). The rich desktop client used by Schedulers has already been built (located under the **Screens** node), and there is no mobile client yet.



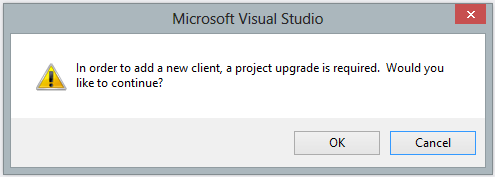
1. Now let’s add a new HTML client to the project. In **Solution Explorer,** open the shortcut menu for the **ContosoMoving** root node and choose **Add Client.**

****

1. In the **Add Client** dialog, enter a unique name, for example **HTMLClient**.

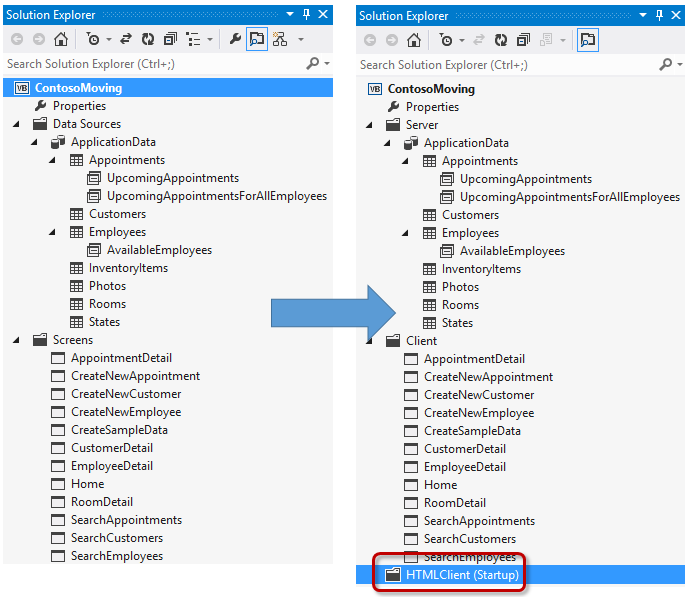


After choosing **OK** in the **Add Client** dialog box, you will get prompt to upgrade the project. This will upgrade the **ContosoMoving** project to include the HTML features. Click **OK** to upgrade.



After upgrade, the new client appears in **Solution Explorer**. Notice the **Data Sources** node is now named **Server**. Desktop client is now under the **Client** node. The newly added HTML client is under **HTMLClient** node.

**HTMLClient** is also automatically designated to be the startup client. The startup client setting simply specifies which client launches when you press **F5**. This can be changed at any time by opening the shortcut menu for a client node in **Solution Explorer** and choosing **Set as StartUp Client**.



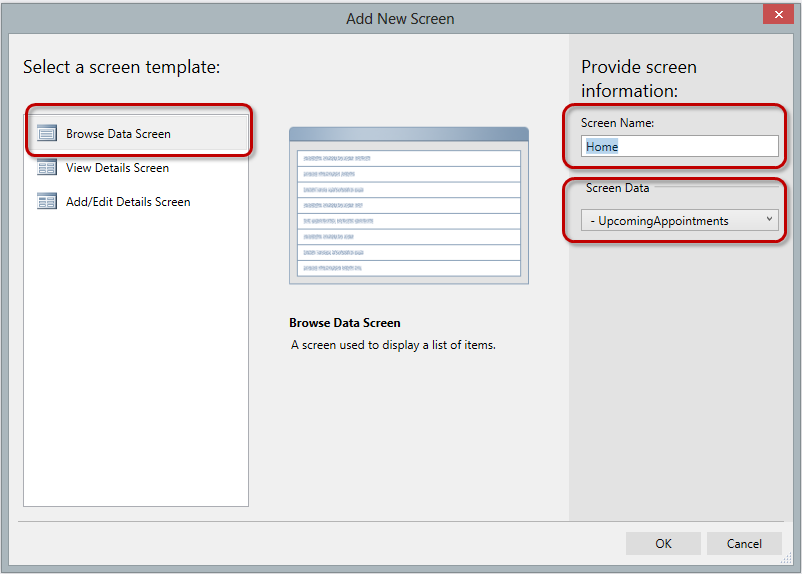
# Step 2: Define Core Screens and App Navigation

Now let’s add some screens for our new mobile client. For those who have used LightSwitch before, the next set of steps should feel very familiar.

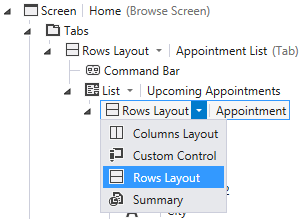
## Create the Home Screen

Let’s make our home screen display a list of upcoming appointments for the currently logged-in user.

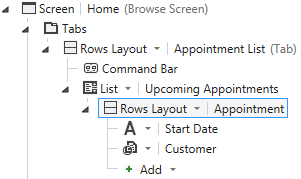
1. In **Solution Explorer**, open the shortcut menu for the **HTMLClient** node and choose **Add Screen**.The **Add New Screen** dialog box opens.
2. In the **Add New Screen** dialog box, specify the following:
   * Screen Template: **Browse Data Screen**
   * Screen Name: **Home**
   * Screen Data: **UpcomingAppointments** (this query is already defined as part of the Server project – it filters for upcoming appointments based on the currently logged-in user).



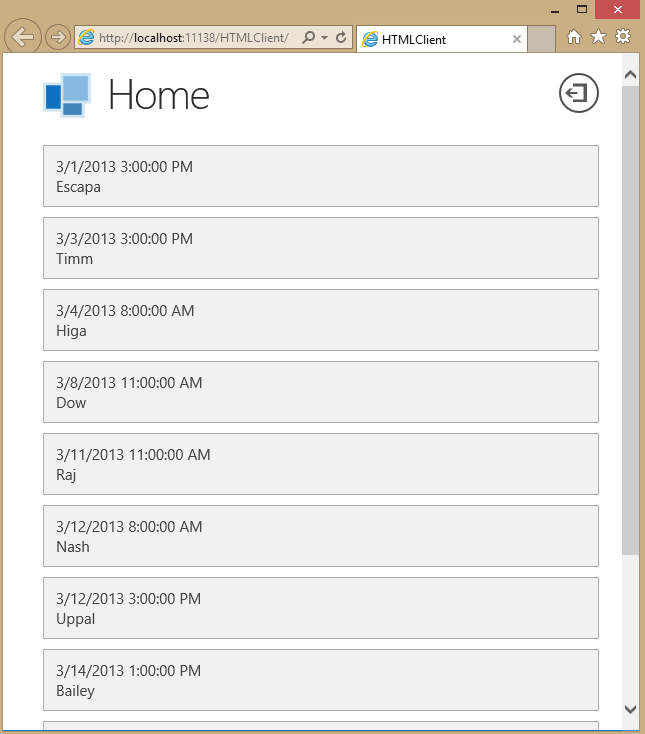
1. Let’s make some simple modifications to how each appointment is displayed in the list.
2. First, change the **Appointment** item’s control to use **Rows Layout**.



1. Secondly, let’s opt to display only the **Start Date** and **Customer** for each appointment item by deleting the other fields. Screen Designer should now look like the following:



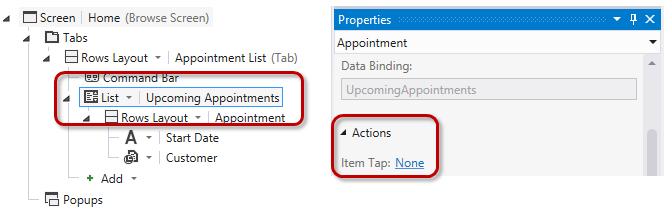
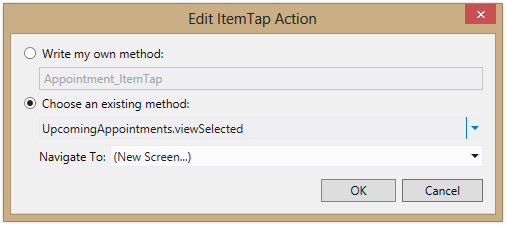
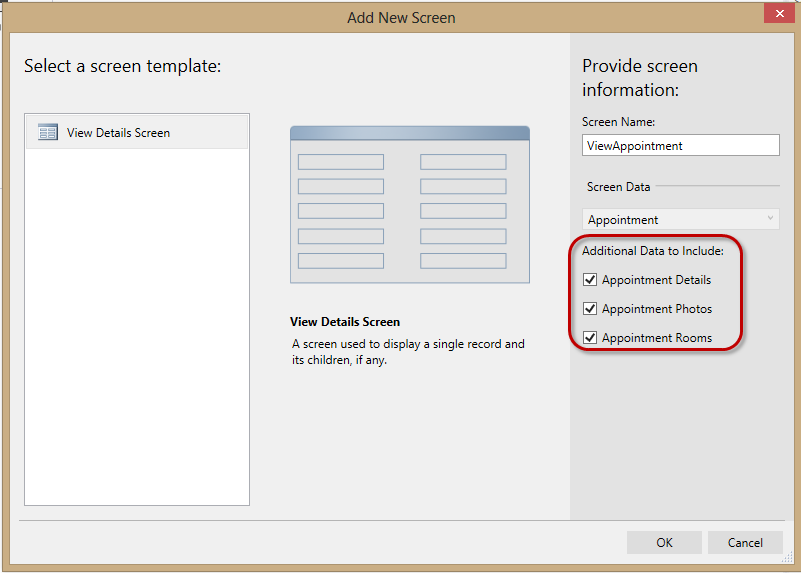
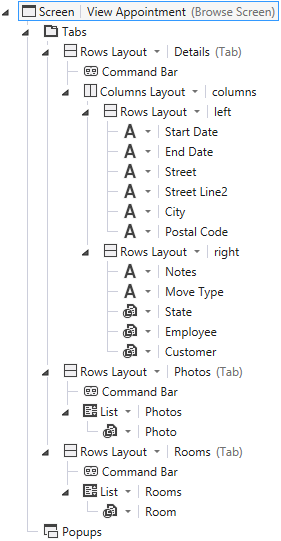
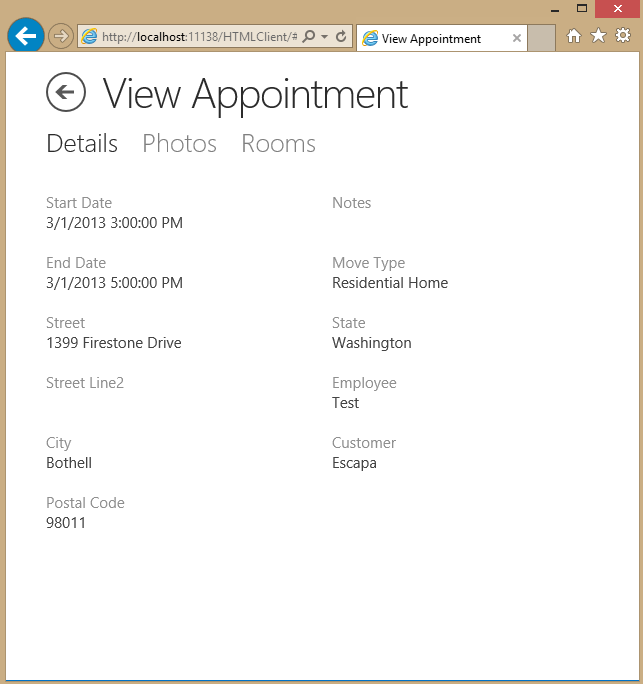
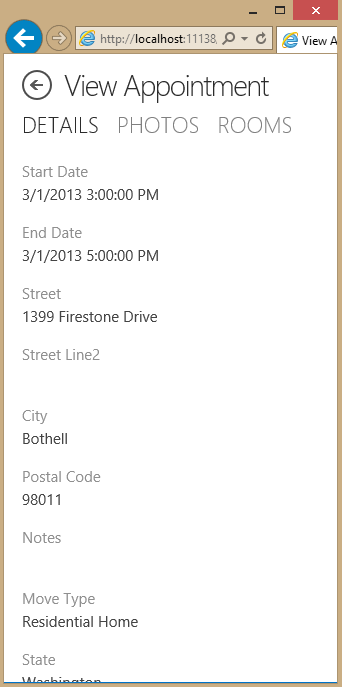
1. Press **F5** to run the application. The default browser will launch and the home screen of the **HTMLClient** will look something like the screenshot below. Later on, we will further customize the home screen.



1. Exit the debug session by closing the browser.

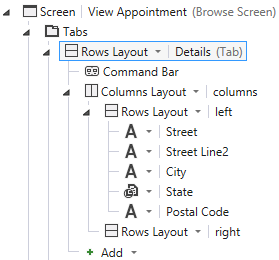
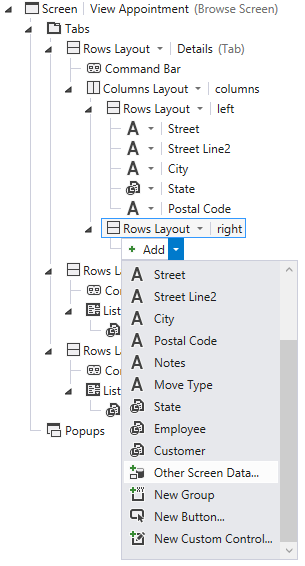
## Create the Appointment Details Screen

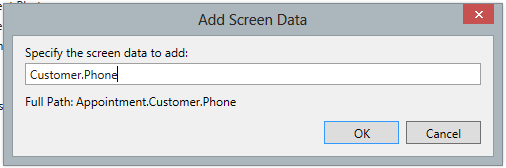
The appointment detail screen is the nucleus of the application—it’s where the inventory specialists will spend most of their time.

1. In the screen content tree, choose the **Upcoming Appointment** List control.
2. In the **Properties** window, choose the **Actions**, **Item Tap** hyperlink to configure what happens when a list item is tapped – the **Edit ItemTap Action** dialog box will appear.   
   
3. We would like to create a view screen for the selected appointment. Therefore, in the **Edit ItemTap Action** dialog box, choose **UpcomingAppointments.viewSelected**. We also need to specify a destination screen. By default, the dialog will guide us to create a new screen. Click **OK**.   
   
4. **Add New Screen** dialog will appear. Check all additional data and click **OK**.   
   
5. Looking at the newly created **ViewAppointment** screen, you will notice that several top-level screen tabs have been created by default (i.e. Details, Photos, and Rooms) to better segment related data.   
   
6. Let’s hit **F5** and run the application! Tap on an appointment to open its **View Appointment** screen. 
7. Resize the browser window. Notice the right column wraps under the left column. The font, size, and paddings also adjust to the new screen real estate. This demonstrates LightSwitch’s ability to support different form factors.   
   
8. Exit the debug session by closing the browser.

# Step 3: Create an Edit Dialog

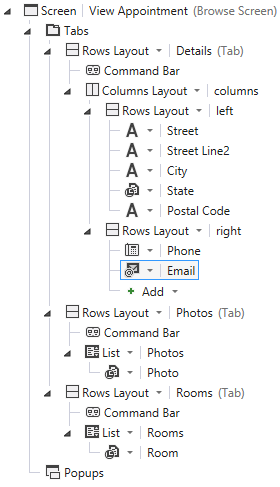
Because receptionists occasionally transcribe information incorrectly, Planning Specialists often verify contact information when they meet a customer and update it as needed. We’ll add support to do this by defining a simple edit dialog.

1. Open the **ViewAppointment** screen.
2. Let’s begin by cleaning up the UI in the main **Details** section (we’ll later add more contextual data in the screen title, so this allows us to simplify the **Details** section):
   1. Remove these fields: **Start Date**, **End Date**, **Notes**, **Move Type, Employee** (since it’s always the current user in this application), and **Customer**.
   2. Move **State** into the left column.   
      
   3. In the right column, add the Customer’s phone number and email:
      1. Choose **Other Screen Data** from the **Add** drop-down.  
         
      2. Type in the name of the field to add: **Customer.Phone.**

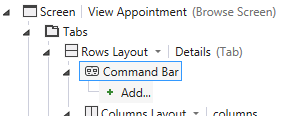
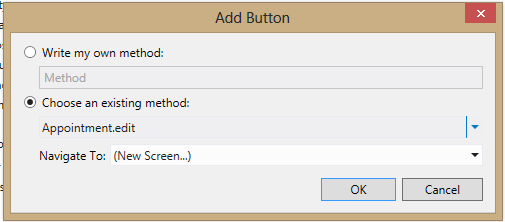
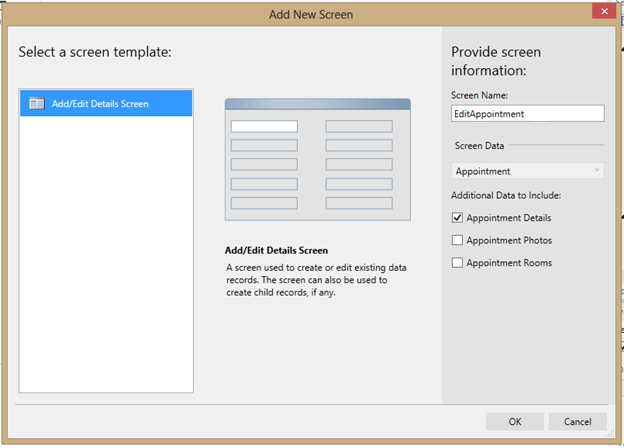


* + 1. Repeat with **Customer.Email.**

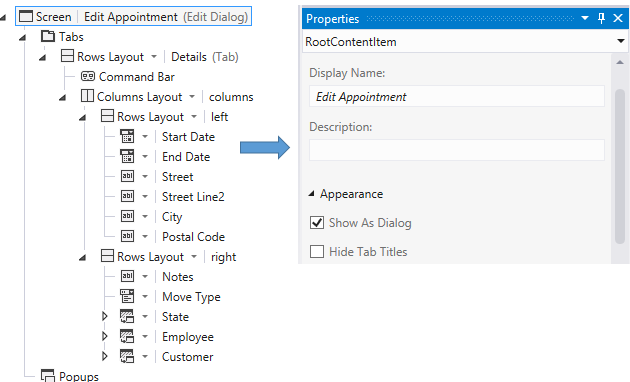
The Screen Content tree should now look something like this:

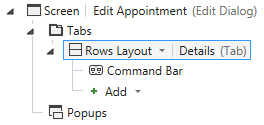


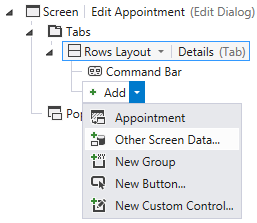
Now it’s time to create our edit dialog. In LightSwitch, dialogs are just screens, visualized in a different way. In this section, we will create a screen (for editing customer information), visualized as a modal dialog.

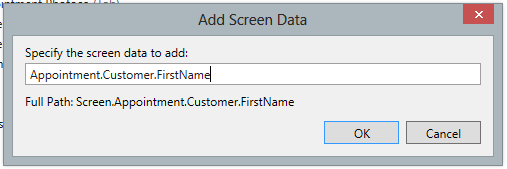
1. We first create a button used to launch the edit dialog. Expand **Command Bar** under **Details** tab. Click **Add**...  
   
2. In the **Add Button** dialog, select **Appointment.edit**. Click **OK**. The dialog will guide us to create a new screen.   
   
3. **Add New Screen** dialog will appear. Name the screen **EditAppointment** and click **OK**.   
   

Notice in the newly created screen, screen’s **Show As Dialog** property is checked. This is to indicate that the screen should be visualized as a modal dialog. By default, **Add/Edit Details Screen** template has this property checked.

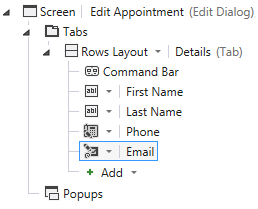


1. In the newly created screen, delete columns node.   
   
   1. Gesture to add **Other Screen Data**, and enter in **Appointment.Customer.FirstName** in the resulting dialog**.**

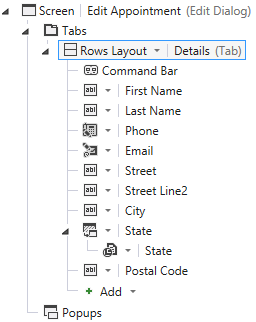


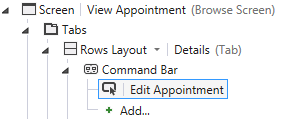


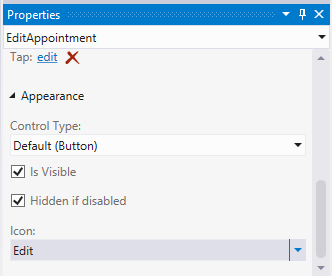
* 1. Repeat the above with customer’s **LastName**, **Phone**,and **Email**.

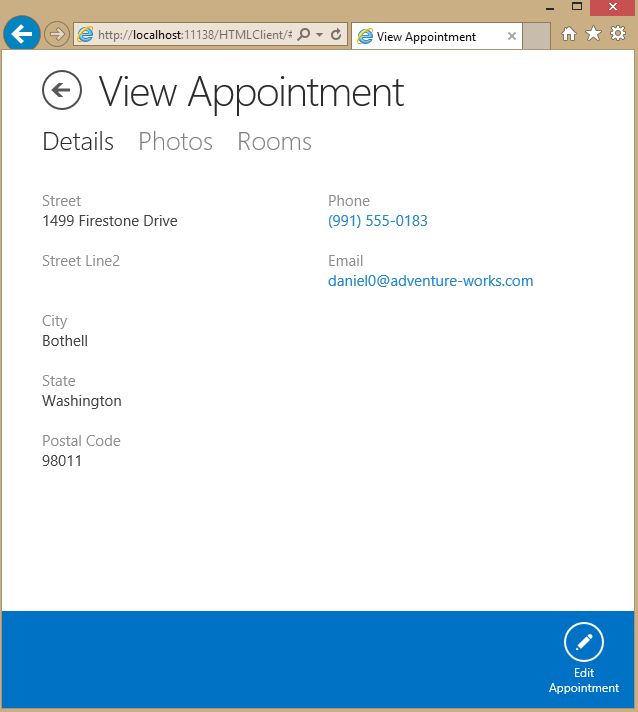
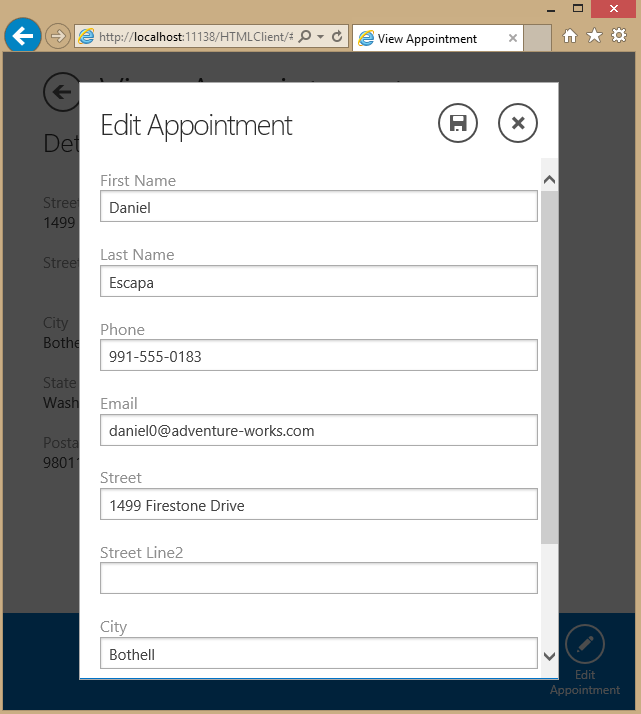


* 1. Add **Street**, **StreetLine2**, **City**, **State**, and **Postal Code** by dragging them from the **Appointment** sidebar into the dialog, or by adding them as above. The resulting dialog should look like this:



1. Open **ViewAppointment** screen, select **Edit Appointment** button we added earlier.   
   

In **Properties**, change **Icon** property to **Edit**.  


1. Press **F5** and run the application. Select an appointment. Notice the Details tab has a command bar button with an edit icon.   
   
2. Tap the **Edit Appointment** button to invoke the dialog – note the Save and Cancel behavior is provided as a built-in capability. Edit the address and click the Save button.   
   
3. Exit the debug session by closing the browser.

# Step 4: Customize the UI

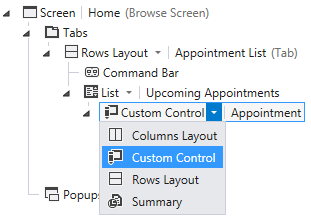
Up until now we’ve mostly kept default UI and we have a functional app, but the UI can definitely be improved. Let’s start addressing some of those improvements here to learn how your knowledge of HTML, CSS, and JavaScript can be applied to customize and extend your application’s UI.

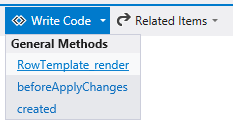
## Custom Formatting using Existing JavaScript Libraries

Let’s do two things: format the appointment date/time to be more legible, and include the customer’s name and phone number.

We can customize how each list item is rendered by defining a custom control for the list item.

1. Open the **Home** screen.
2. Select the **Appointment** item, and change its control from **Rows Layout** to **Custom Control**.



1. Choose the **Write Code** drop-down list and choose **RowTemplate\_render**. This will open the JavaScript code editor.  
   
2. Insert the following code in the **RowTemplate\_render** function. Again, the code in this example is standard JavaScript – it defines HTML content and adds it to an appointment’s div element. In this function, **contentItem** represents the **Appointment** data item, so we can refer to the appointment’s **StartDate** and **FirstName** fields by contentItem.value.StartDate and contentItem.value.FirstName respectively.

myapp.Home.RowTemplate\_render = function (element, contentItem) {

var itemTemplate = $("<div> </div>");

var title = $("<h3>" + moment(contentItem.value.StartDate).format("ddd, MMM Do, h:mm") + "-" + moment(contentItem.value.EndDate).format("h:mma") + "</h3>");

var subTitle = $("<span>" + contentItem.value.Customer.FirstName + " " + contentItem.value.Customer.LastName + " - " + contentItem.value.Customer.Phone + "</span>");

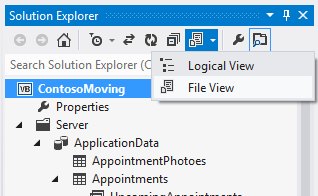
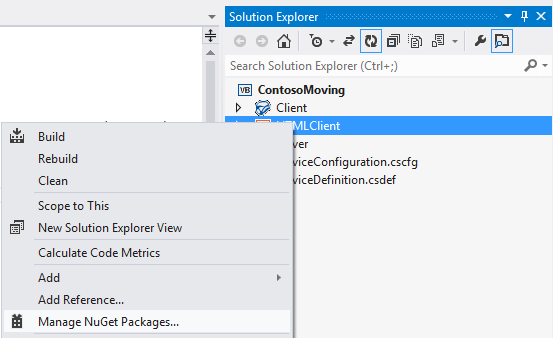
title.appendTo($(itemTemplate));

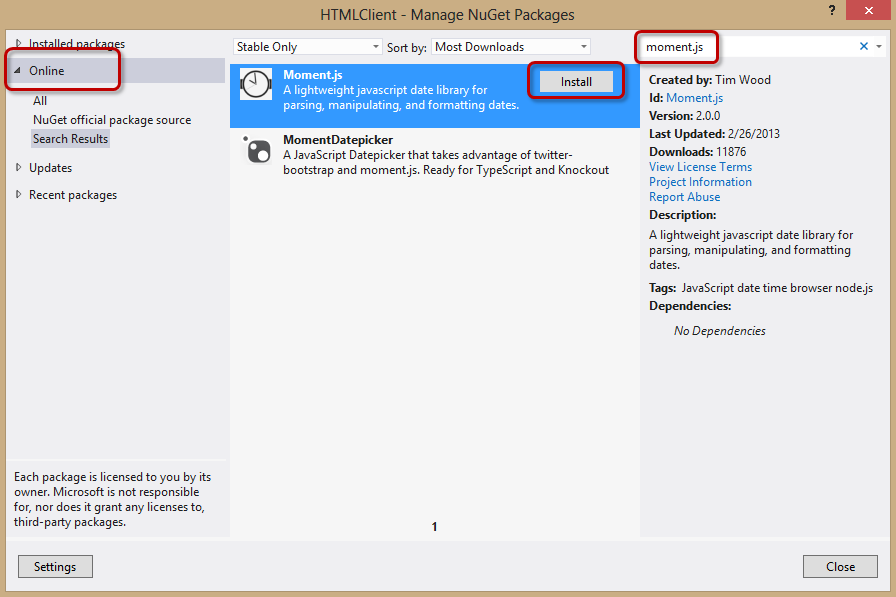
subTitle.appendTo($(itemTemplate));

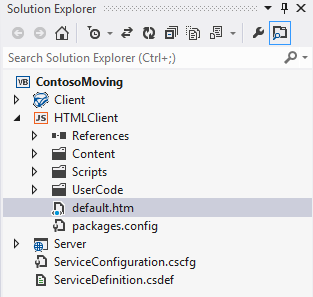
itemTemplate.appendTo($(element));

};

We need to do one more step – there are many JavaScript libraries available on the web that help parse and format dates. The code above that formats the appointment’s **StartDate** uses a JavaScript library called **Moment.js** that we’ll need to include in our project. Any of the available JavaScript date libraries could be used for this example, but let’s use **Moment.js** as it will give us the opportunity to use the **NuGet** management features inside Visual Studio 2012.

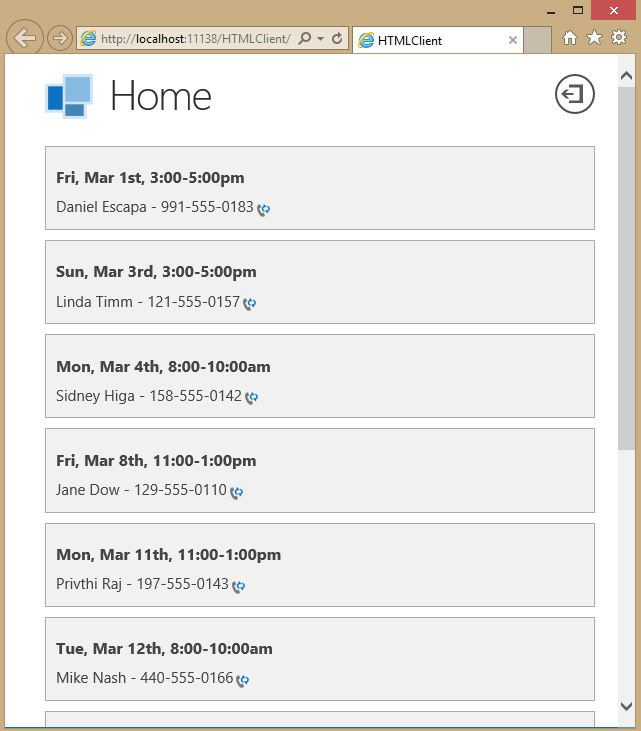
1. In **Solution Explorer**, select **ContosoMoving** node. Use the menu to switch to **File View.**
2. Open the shortcut menu for the **HTMLClient** project and select **Manage NuGet Packages.**
3. In the **Manage NuGet Packages** dialog box, do the following:
   1. Choose the **Online** category (on the top-left).
   2. Enter “moment.js” in the Search textbox (top-right). Search results will display the NuGet package for Moment.js.
   3. Choose the **Install** button for Moment.js.



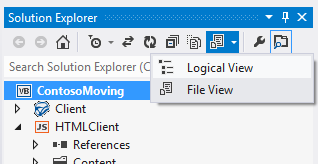
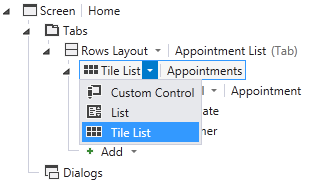
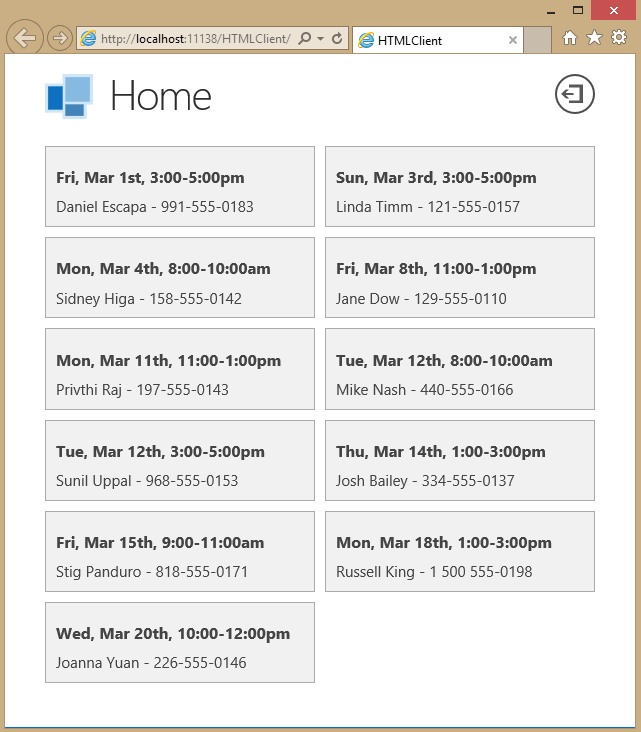
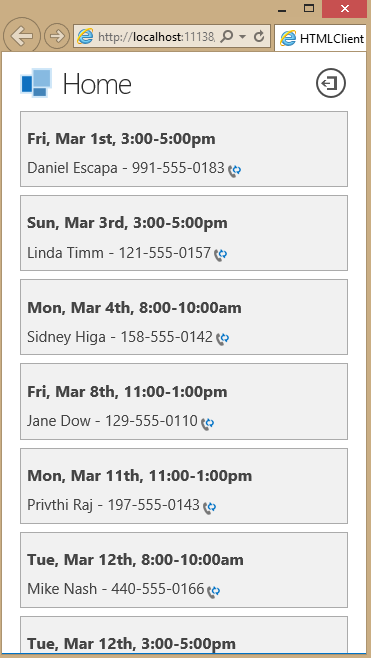
1. In **default.htm** (under HTMLClient node), reference moment.js using the script tag (you can insert it as the last of the script reference tags).  
   

<script type="text/javascript" src="Scripts/moment.js" charset="utf-8"></script>



1. Hit **F5** to run the app! We have just customized the list item. Hopefully you will have observed how standard HTML, JavaScript, and CSS – for which there is a vibrant community providing many resources – can be used to easily extend LightSwitch UI.   
   
2. Exit the debug session by closing the browser.

## Using Tile List

1. In **Solution Explorer**, select **ContosoMoving** node. Use the menu to switch back to **Logical View**.  
   
2. Open **Home** screen.
3. Change the **Appointments** from **List** to **Tile List** control.  
   
4. Hit **F5** to run the app! The home screen now displays an array of tiles based on the size of your browser.   
   
5. Adjust the size of the browser window. Notice the list re-tiles based on the form factor.  
   
6. Exit the debug session by closing the browser.

## Update the Appointment Screen to have a Dynamic Screen Title

Let’s place contextual information in the screen title since we removed the **Start Date** and **Customer Name** fields from the **Details** tab.

1. Open the **View Appointment** screen.
2. Choose the **Details** tab in the Screen Content tree.
3. Open the **Write Code** menu and choose **Details\_postRender**. Add the code below:  
     
   myapp.ViewAppointmentDetail.Details\_postRender = function (element, contentItem) {

contentItem.dataBind("value.Appointment.Customer.FirstName", function () {

formatAppointmentScreenTitle(contentItem);

});

contentItem.dataBind("value.Appointment.Customer.LastName", function () {

formatAppointmentScreenTitle(contentItem);

});

function formatAppointmentScreenTitle(contentItem) {

contentItem.screen.details.displayName = contentItem.value.Appointment.Customer.FirstName + " " +

contentItem.value.Appointment.Customer.LastName + " " +

moment(contentItem.value.Appointment.StartDate).format("h:mma") + " - " +

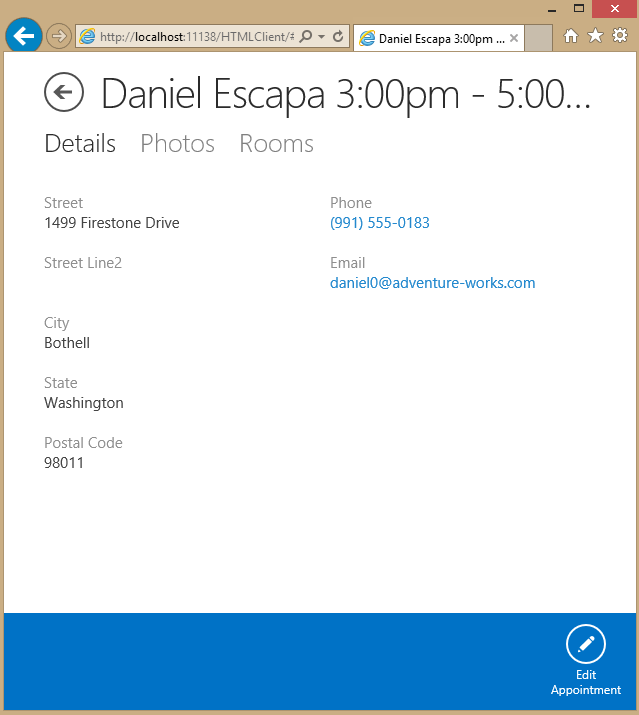
moment(contentItem.value.Appointment.EndDate).format("h:mma");

};

};

This example makes use of a LightSwitch API: **dataBind()**. This API provides a simple, yet powerful, means to data bind screen content to functions. In the code sample above, this means that whenever the data for **FirstName** or **LastName** changes, the function **formatAppointmentScreenTitle()** will be reevaluated, thereby updating the screen title.

1. Hit **F5** to run the app. Tap on an appointment to see the new screen title. If we change the customer’s name in the **Edit Appointment** diaolog, notice how the screen title updates dynamically.



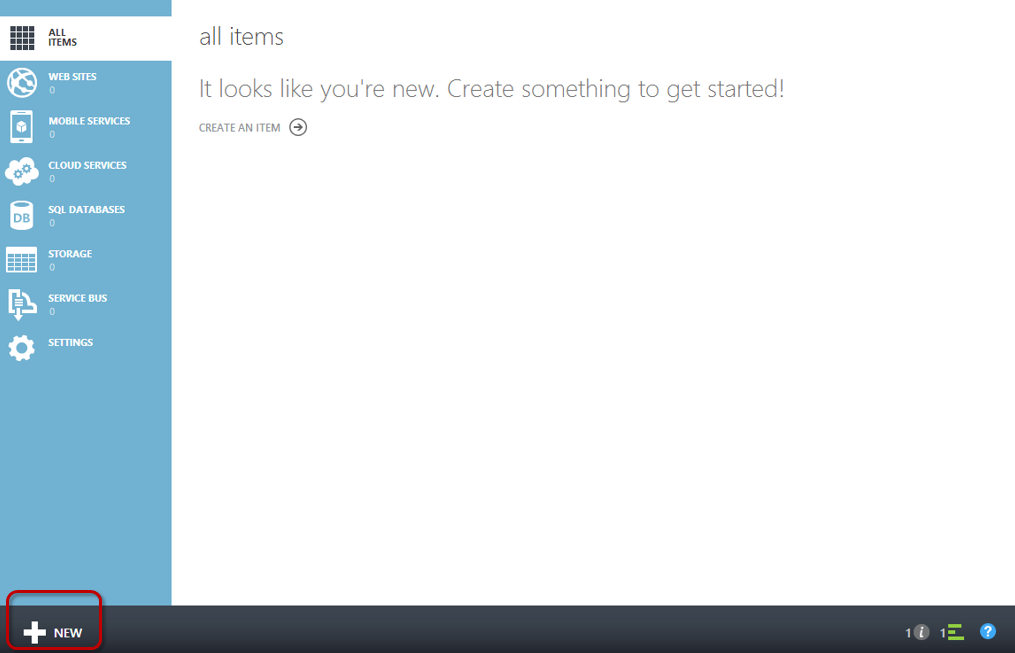
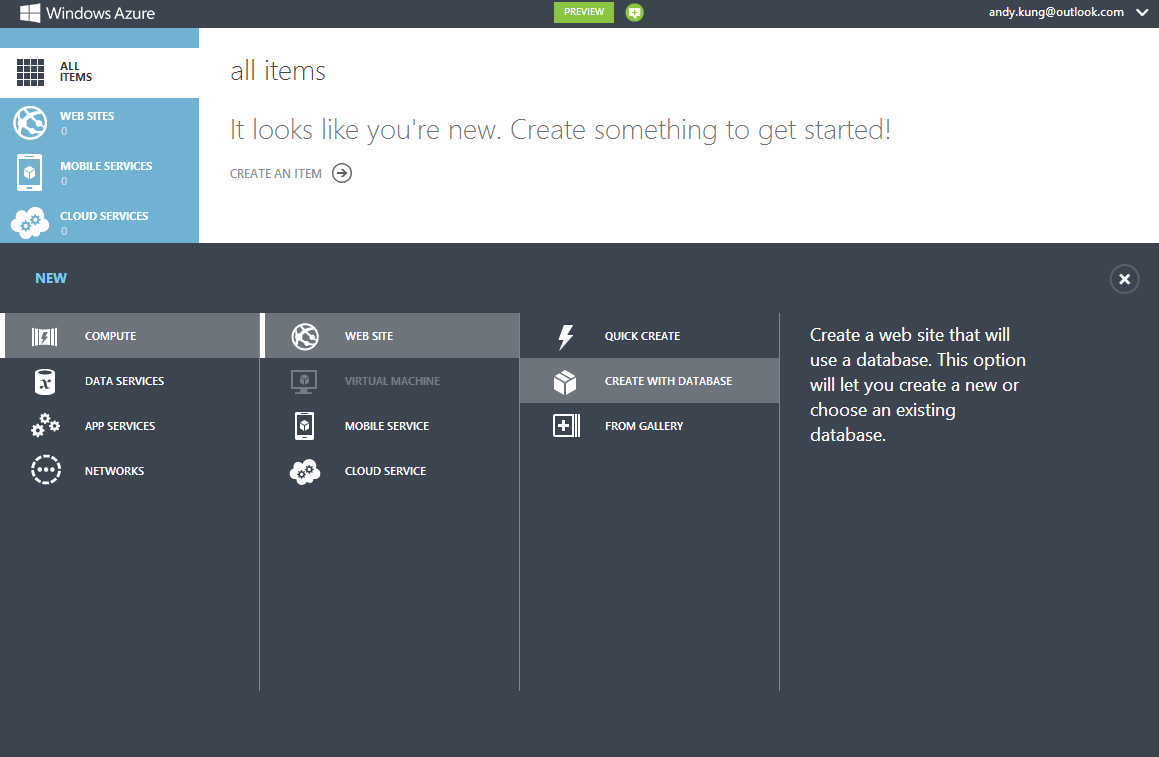
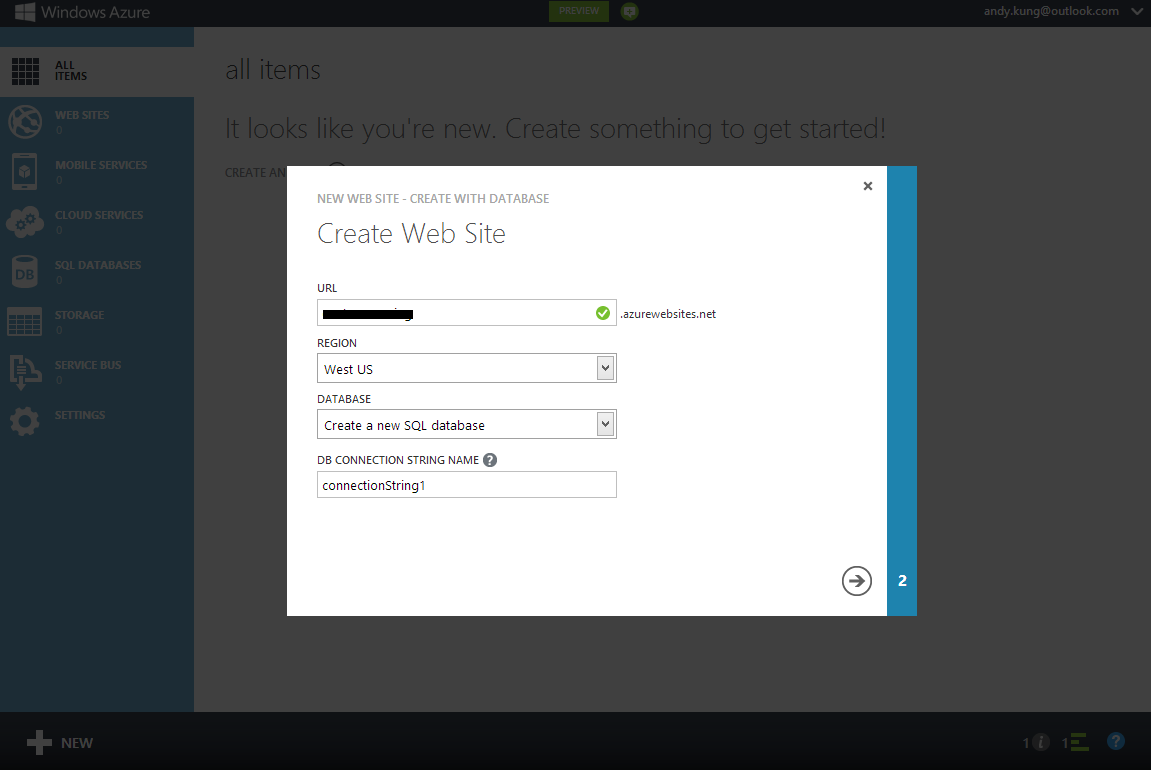
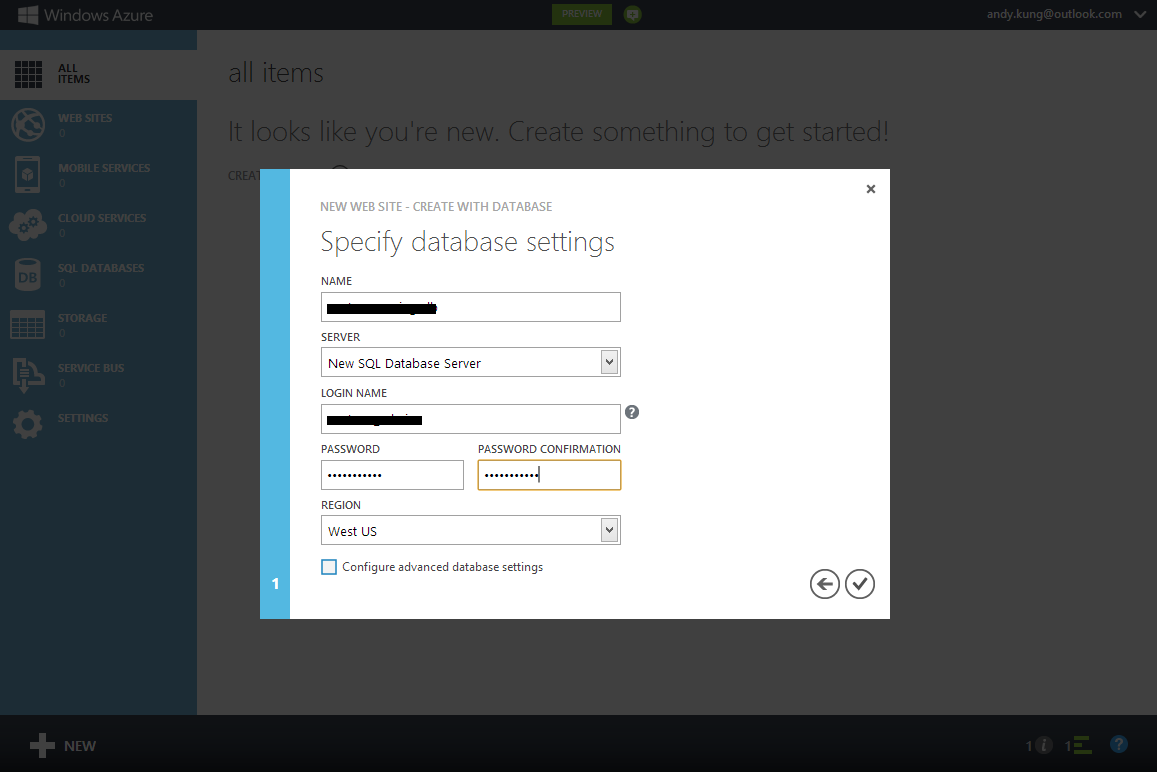
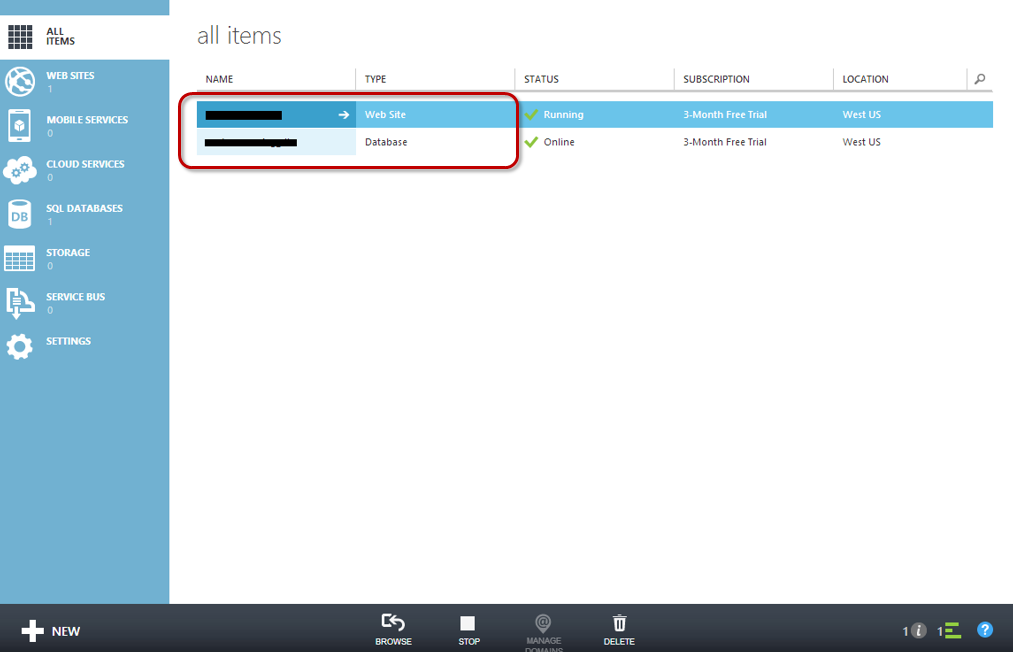
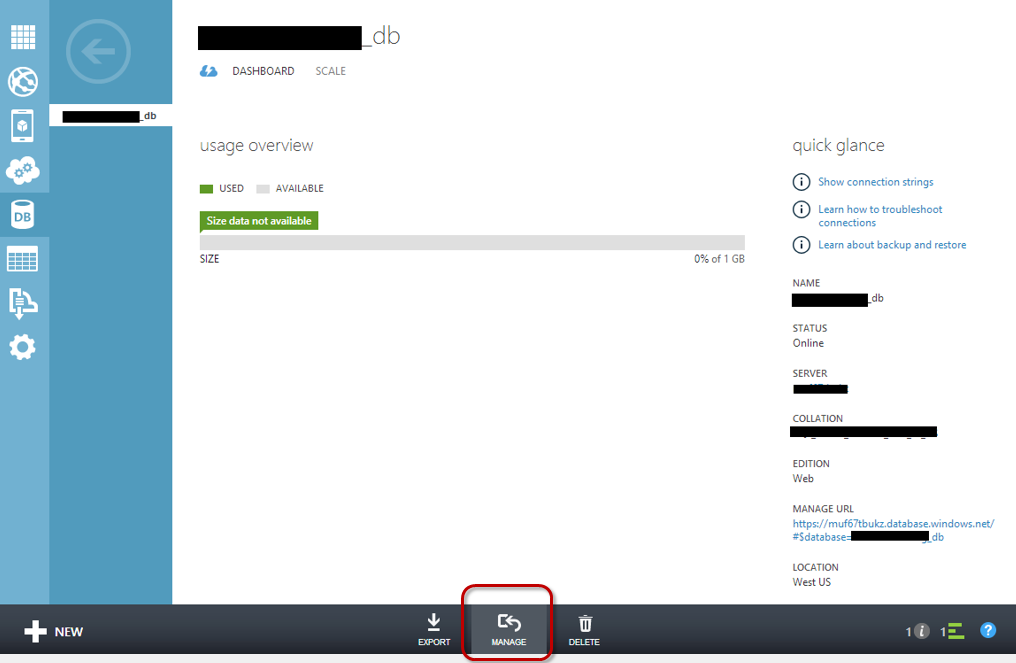
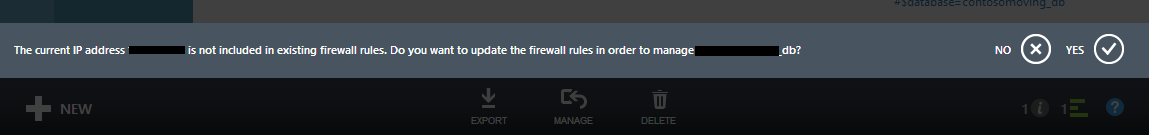
1. Exit the debug session by closing the browser.

# Step 5: Test the Application on a Tablet Device

So far we have run the application in debug mode and seen it in the desktop browser. It’d be nice to publish the application and reach it from a mobile device’s browser since that’s what its designed to be used on.

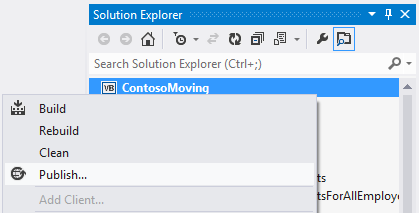
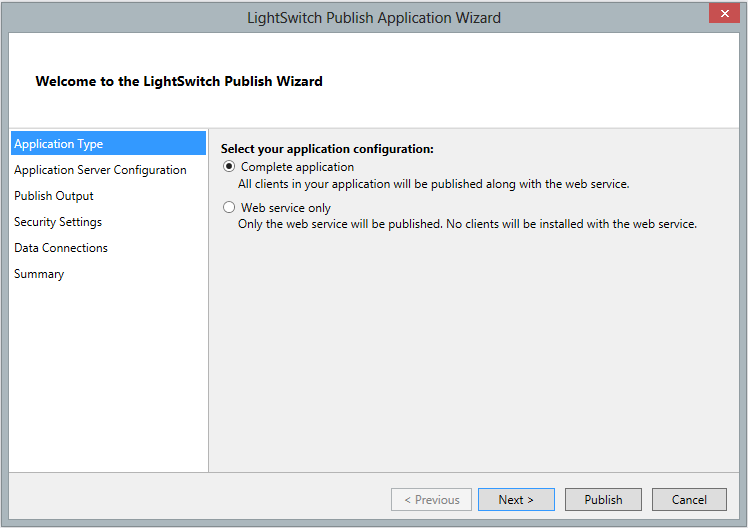
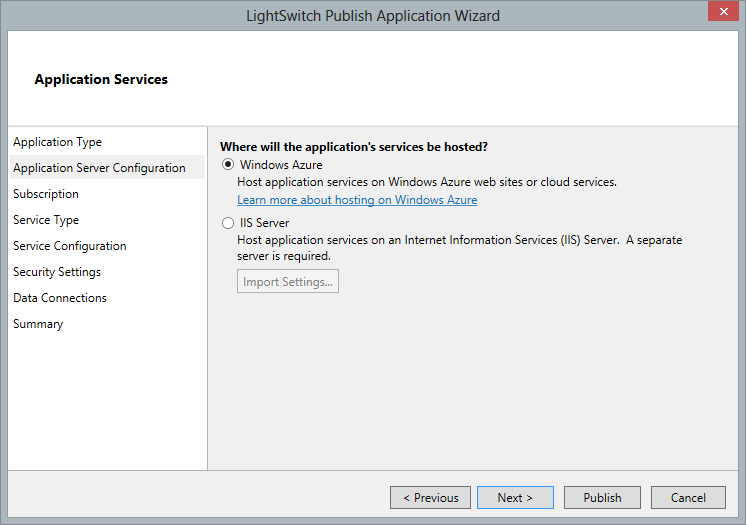
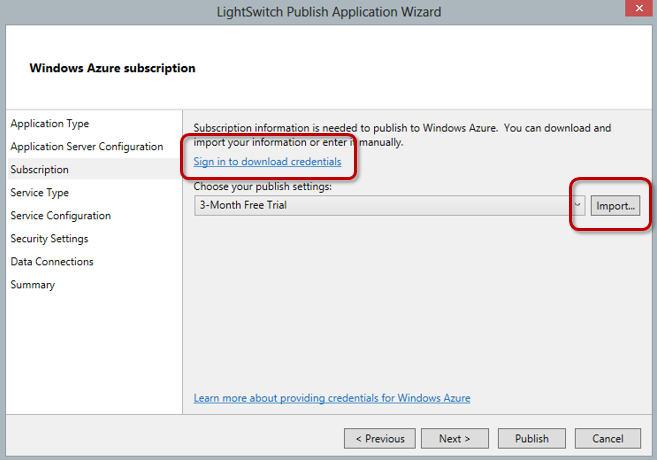
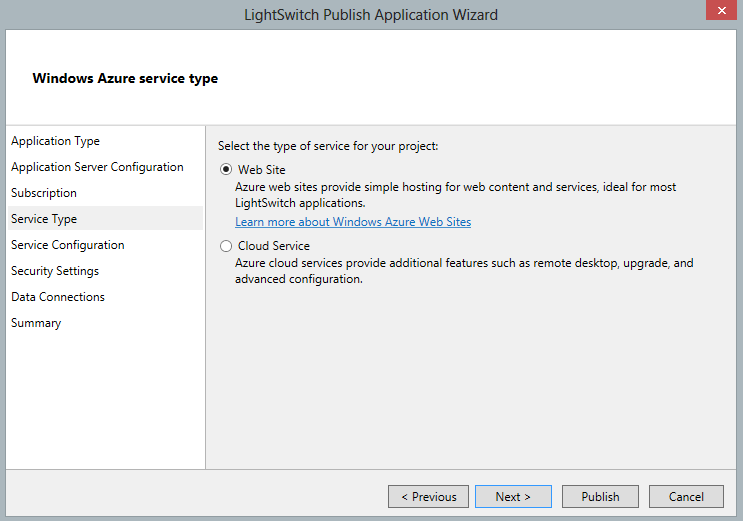
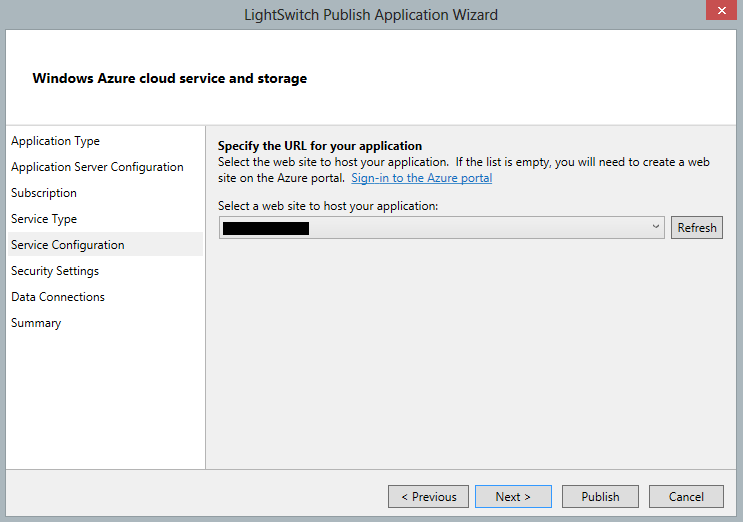
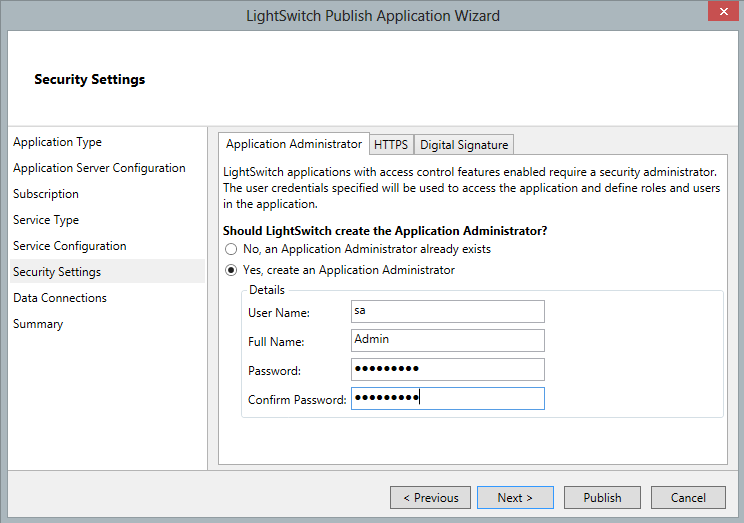
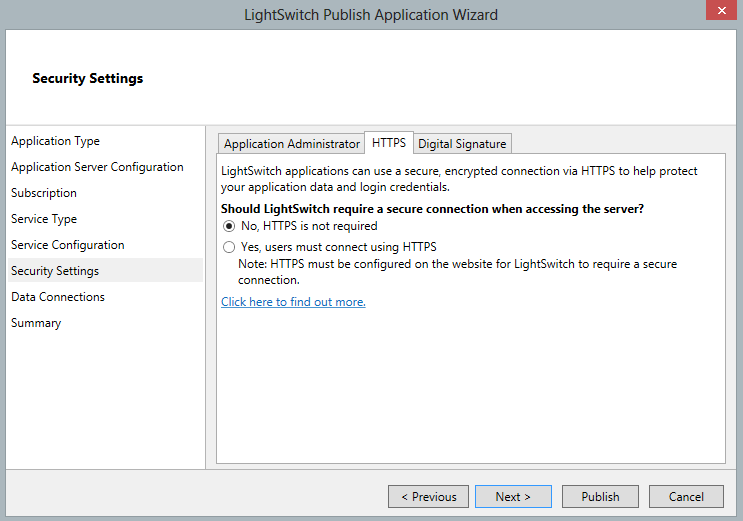
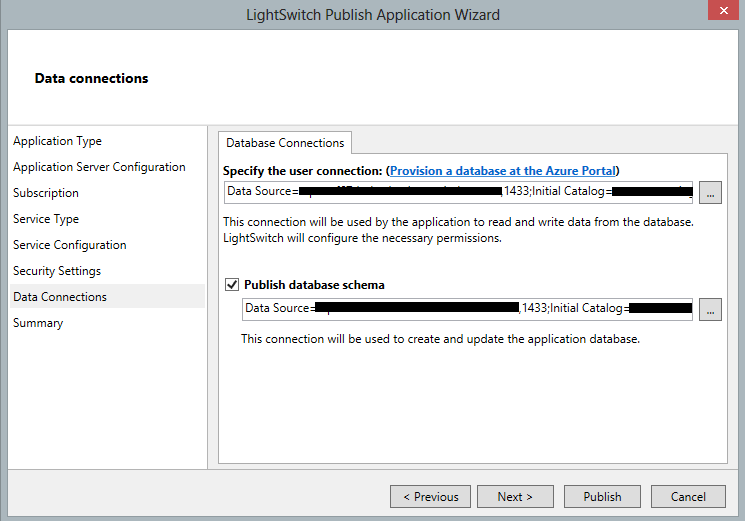
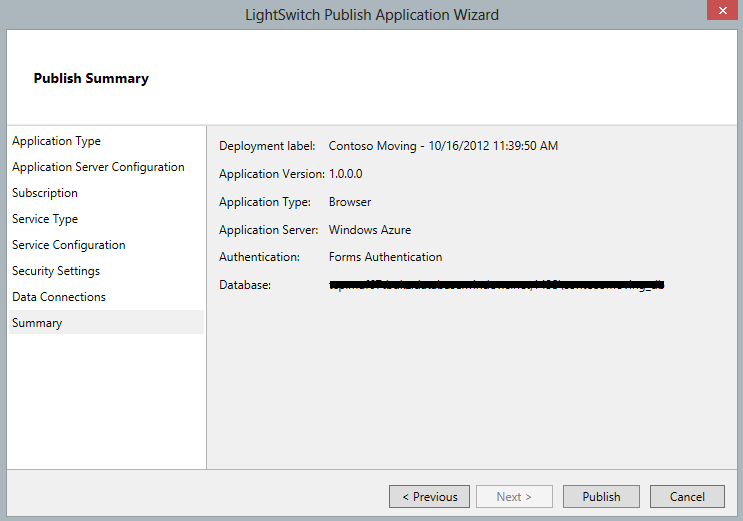
Let’s publish it to the cloud!

## Windows Azure

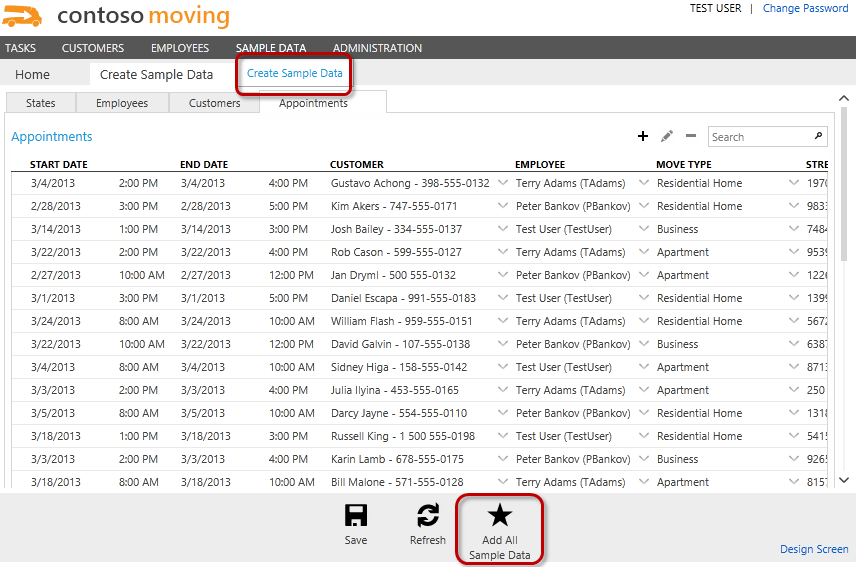
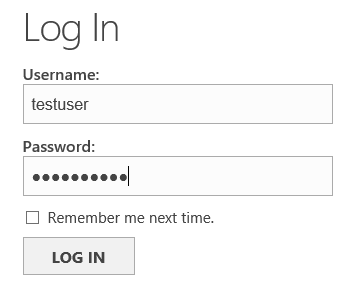
1. Before we start, make sure you have a [Windows Azure account](http://www.windowsazure.com/). If you don’t, we encourage you to take advantage of the [90 day free trial](https://www.windowsazure.com/en-us/pricing/free-trial/). It’s easy to sign up!
2. Once you have an acount, the first thing we need to do is to provision a web site and database for the Contoso Moving application. Log in to your [Windows Azure Portal](https://manage.windowsazure.com/).   
   
3. To create a web site and database, click **NEW** at the bottom. Select **COMPUTE**, **WEB SITE**, **CREATE WITH DATABASE**. 
4. Create a **URL** of your choice. This is the URL you will be testing with on your mobile device. Pick a **region** based on where you are. Select **Create a new SQL database**.   
   
5. Next, give your database a **name**. Select **New SQL Database Server**. Create a **login name** and **password**. Pick a **region** based on where you are.   
   
6. That’s it! Your Azure web site and database is now created! You can see them on the **ALL ITEMS** tab in the **Windwos Azure Portal**.   
   
7. One last thing. You must open the firewall on your database to let LightSwitch publish to that database remotely. By default, your Azure database will not allow for remote management. You need to enable remote management of your database from your IP address: from the computer that will publish to the web site, select your database in the portal and click **MANAGE** at the bottom.   
   
8. Windows Azure will ask you if you want to enable management of the database from your current IP. Select **YES**.   
   

## Publish

We’re now ready to publish Contoso Moving to the cloud!

1. In **Solution Explorer**, open the shortcut menu for the root node and select **Publish**.  
   
2. The **Publish Wizard** will guide you through the publish steps. We are going to publish the **Complete application**. Click **Next**.   
   
3. Select **Windows Azure** to host your application service. Click **Next**.   
   
4. Click **Sign in to download credentials**. This will prompt you to log in to your **Windows Azure** account to download a settings file. **Import** it into the publish wizard. Once imported, these settings are saved, encrypted in your profile. Other users in VS cannot see or use these credentails. You may delete your download settings file. Click **Next**.   
   
5. Select **Web Site**. Click **Next**.   
   
6. Choose which website you want to publish to. Select the **Azure web site** you previsioned in the dropdown menu. Click **Next**.   
   
7. Since *Contoso Moving* uses **Forms** authentication. You must supply an admin account so you can log into the web site when it’s published. Remember this credentail. We will need it later.  
   
8. Click **HTTPS** tab. You will be given the option to choose whether HTTPS is required for your application. It is recommended that you use HTTPS for your LightSwitch application. In which case you must configure your web site to require a secure connection. In our walkthrough, we will keep it simple and select **No, HTTPS is not required**. Click **Next**.   
   
9. The final step in the process is to specify the database connections. Click **Next**.   
   
10. The last page in the wizard lets you review your information. Click **Publish**!  
    
11. After publish succeeds, you will be able to access both the rich desktop client and HTML client via a web browser.
    * Desktop client: **[name]. azurewebsites.net/Client**
    * HTML client: **[name].azurewebsites.net/HTMLClient**

Since there is no data currently in the Azure database, let’s first log in to the **desktop client** with the admin account you supplied during publish process to add some data.   

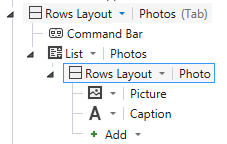
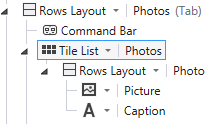
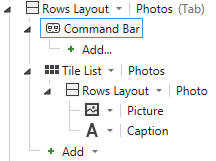
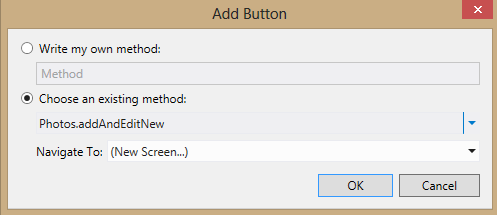
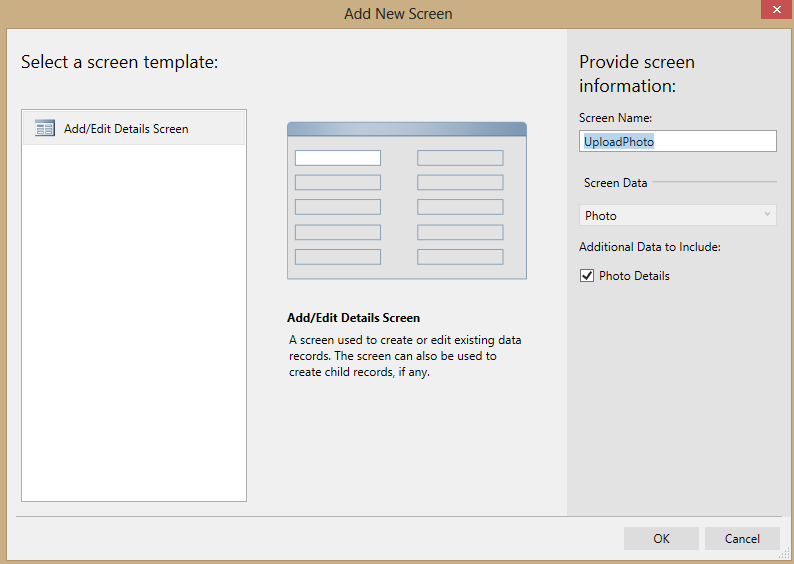
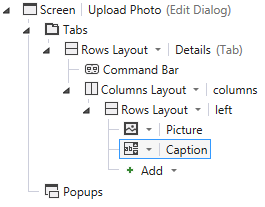
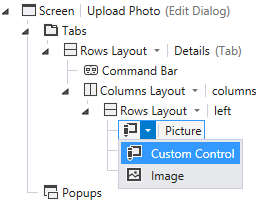

1. Use **Add All Sample Data** feature to generate some sample data like how we started the walkthrough.   
   
2. On a mobile device, open the web browser and go to **[name].azurewebsites.net/HTMLClient**. Login with username: **testuser**, password: **pass@word1**. This is one of the sample data we generated. See the HTML client in action!  
   

# Step 6: Using a Device’s Built-in Camera to Shoot and Upload Photos

Let’s continue improving our app! We will give planning specialists the option to use use their mobile device to upload pictures of the home. The intent is that the inventory specialist will take a picture with the device, save it locally, and then upload it via the ContosoMoving Mobile app.

1. In **Solution Explorer,** switch to **File View**. Choose the **HTMLClient**, **Scripts** folder.
2. Open the shortcut menu for the **Scripts** folder and choose **Add, Existing** **Item**.
3. From within the dialog, navigate to the **Sample Resource** folder (provided as part of the sample), and add **image-uploader.js** and **image-uploader-base64-encoder.aspx**.
4. Open **default**.htm and add a reference to **image-uploader.js** at the end of the script block.

<script type="text/javascript" src="Scripts/image-uploader.js" charset="utf-8"></script>

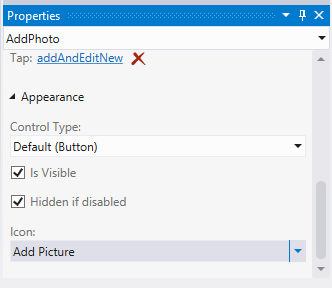
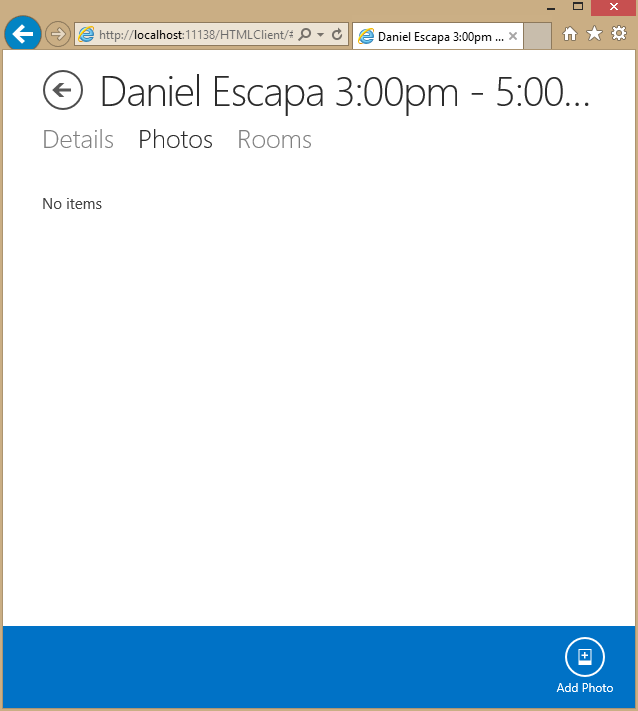
1. In **Solution Explorer**, switch back to **Logical View** and open the **ViewAppointment** screen.
2. In the **Screen Designer**, change the **Photo** summary field to a **Rows Layout**.   
   
3. Change **Photos** node to **Tile List**. In **Properties** window, change the **Picture**’s width to **Stretch to Container** and height to **150**.   
   
4. Expand **Command Bar** under **Photos** tab. Click **Add**...   
   
5. In the Add Button dialog, choose **Photos.addAndEditNew** and click **OK**.   
   
6. In the **Add New Screen** dialog, name the screen **UploadPhoto** and click **OK**.  
   
7. In **AddPhoto** screen, delete right column field.
8. Change **Caption**’s type to **Text Area.**
9. Switch **Picture** field from an **Image** to a **Custom Control**.   
   
10. In the **Properties** window, choose the **Edit Render Code hyperlink** and add the following code:

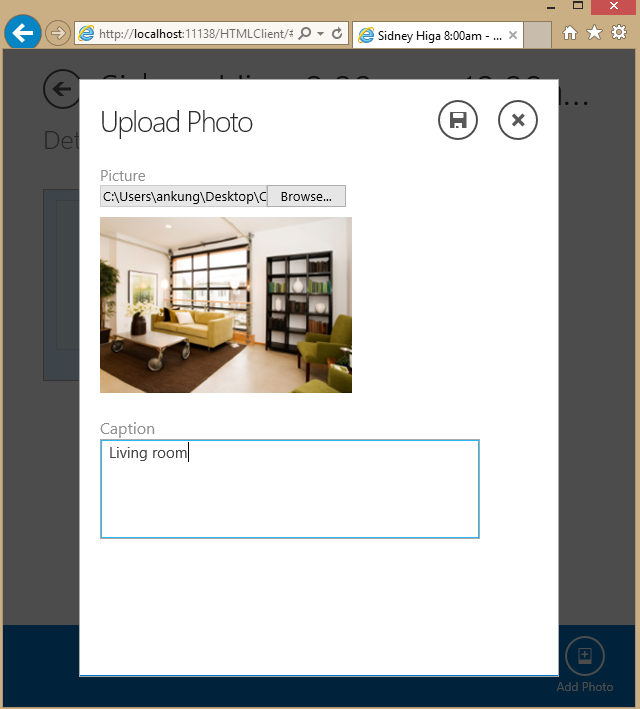
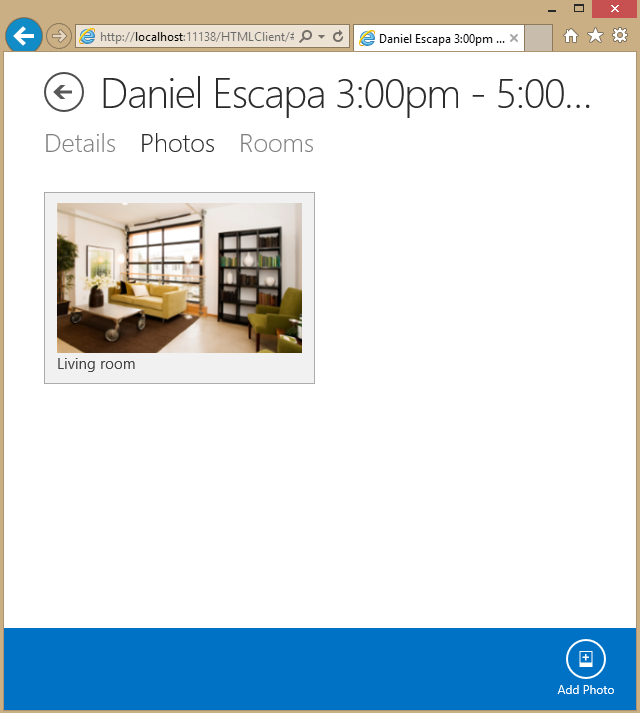
myapp.ViewAppointmentDetail.Picture1\_render = function (element, contentItem) {

createImageUploader(element, contentItem, "max-width: 300px; max-height: 300px");

};

The call redirects all of the heavy-lifting to the **image-uploader.js** file.

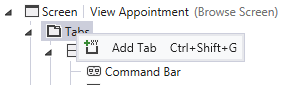
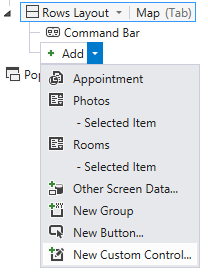
1. Go back to **ViewAppointment** screen. Select the **Add Photo** button we added earlier. In **Properties**, change the **Icon** property to **Add Picture**.  
   
2. Press **F5** to run the app. Select an appointment and switch to the **Photos** tab. Notice the tab has a command bar button with a picture icon.   
   
3. Click **Add Photo** and see the image-uploader in action.

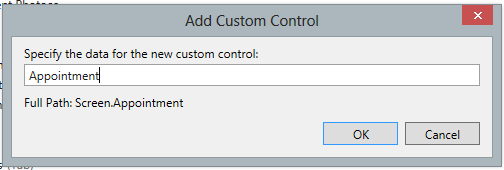
  
  
  
**NOTE: File uploading is not supported by iOS 5’s Safari browser and Windows Phone 8’s IE browser.**

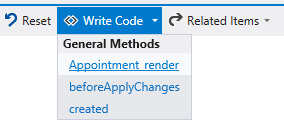
1. Exit the debug session by closing the browser.

# Step 7: Add a Bing Map Custom Control

Now let’s modify the **ViewAppointment** screen to display a map for the current appointment’s address. Similarly, we need to begin by adding some files to our project, and referencing them from the **default.htm** file.

1. In **Solution Explorer**, switch to **File View** and choose the **Scripts** folder of the **HTMLClient** project.
2. Open the shortcut menu for the **Scripts** folder and choose **Add**, **Existing Item**.
3. Navigate to the **Sample Resource** folder (provided as part of the sample) and add **lightswitch.bing-maps.js**.
4. Open the **default.htm** file.
5. Add script references for the **lightswitch.bing-maps.js** file at the **end** of the script block.  
   <script type="text/javascript" src="Scripts/lightswitch.bing-maps.js" charset="utf-8"></script>
6. Add a CDN reference to the Bing Maps control in the **beginning** of the script block.  
   <script type="text/javascript" charset="utf-8" src="http://ecn.dev.virtualearth.net/mapcontrol/mapcontrol.ashx?v=7.0"></script>
7. In **Solution Explorer**, switch back to **Logical View.**
8. Open the **ViewAppointment** screen in the designer.
9. Open the shortcut menu for **Tabs** and choose **Add Tab**. Name the new tab **Map**.  
   
10. Choose **Add**, **New Custom Control**.  
    
11. In the **Add Custom Control** dialog box, specify **Appointment** as the data for the custom control.



1. Use the **Write Code** menu to edit the **Appointment\_render** logic for the control.  
   
2. Add the following utility methods **above** the render function:  
     
   function rebindMap(element, contentItem) {

// Check to make sure we aren't updating the map continuously due to multiple bound values changing.

var now = new Date();

if (now.getTime() - mapLastUpdated.getTime() > 15) {

setTimeout(function () {

updateMap(element, contentItem);

mapLastUpdated = new Date();

}, 20);

}

};

function updateMap(element, contentItem) {

var mapDiv = $("#appointmentMap");

// If we've previously created the map, make sure to clean up the div it was contained in;

// otherwise the Bing map control fails to create properly.

if (mapDiv.length > 0) {

$(mapDiv).remove();

}

mapDiv = $("<div id='appointmentMap' class='msls-hauto msls-vauto' ></div>");

$(mapDiv).appendTo($(element));

mapControl = mapDiv.lightswitchBingMapsControl({

street: contentItem.value.Street,

city: contentItem.value.City,

state: contentItem.value.State.Name,

zipcode: contentItem.value.PostalCode,

mapTypeId: Microsoft.Maps.MapTypeId.road,

height: "400"

});

};

1. Now bind the map control the address fields on the appointment by adding the code to the **Appointment\_render** method:  
     
   myapp.ViewAppointmentDetail.Appointment\_render = function (element, contentItem) {

updateMap(element, contentItem);

mapLastUpdated = new Date();

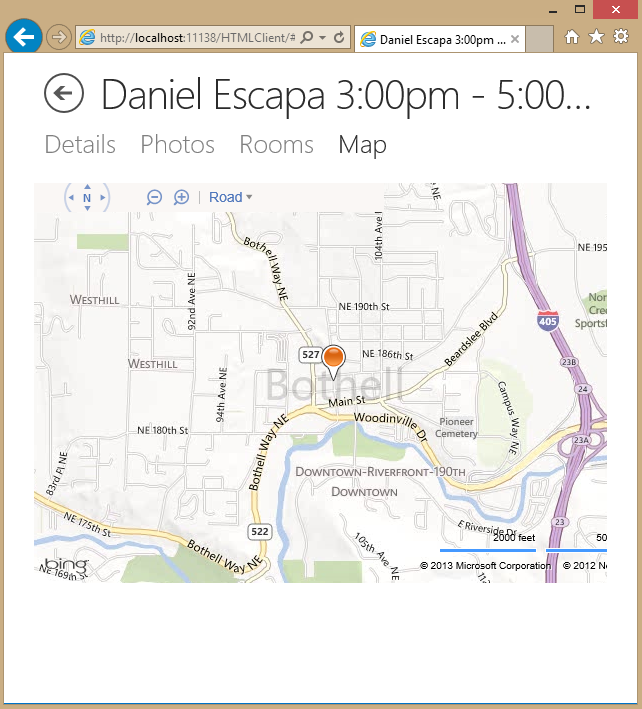
contentItem.dataBind("value.Street", function () { rebindMap(element, contentItem); });

contentItem.dataBind("value.City", function () { rebindMap(element, contentItem); });

contentItem.dataBind("value.Zip", function () { rebindMap(element, contentItem); });

contentItem.dataBind("value.PostalCode", function () { rebindMap(element, contentItem); });

};

1. Press **F5**. After the application loads, tap on an appointment, and then tap the **Map** section.  
   
2. Exit the debug session by closing the browser.

# Step 8: Republish the App and Test it on a Tablet Device

1. In **Solution Explorer**, open the shortcut menu for the **Contoso Moving** project and choose **Publish**.
2. The **Publish Wizard** remembers your existing configuration and automatically shows the Summary page. Choose the **Publish** button.
3. Check out your changes on a mobile device!

# Step 9: Brand the Application

Finally, let’s see how we can brand the application with the corporate logo and colors.

## Add an application title

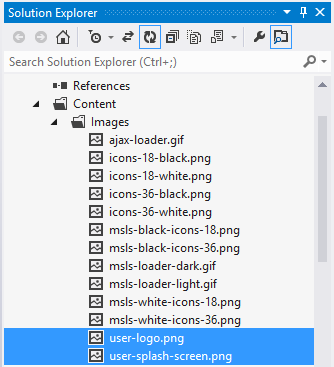
The default title for the HTML client is the name of the client. In this case **HTMLClient**. To customize it:

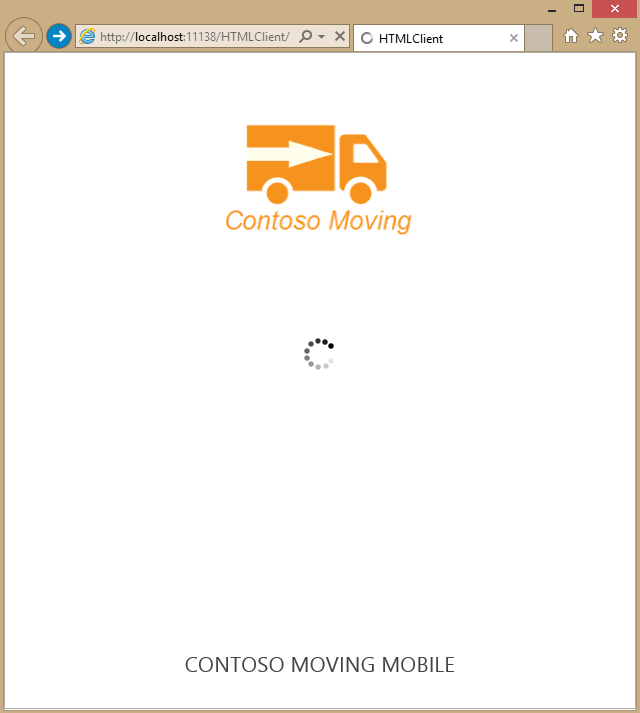
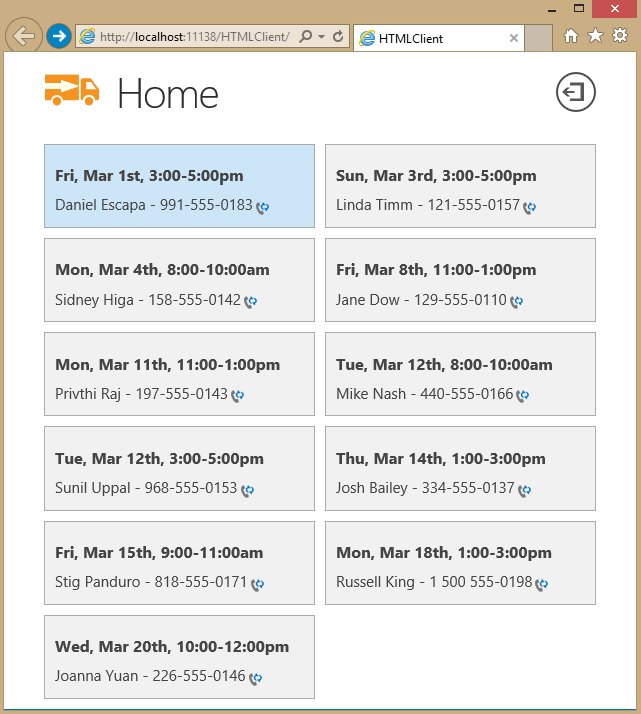
1. In the **Solution Explorer**, switch to **File View**. Under the **HTMLClient** node, open **default.htm**.
2. Modify <title>HTMLClient</title> near the top of the file to include a title for your app. For example, <title>Contoso Moving Mobile</title>.  
   

## Add an icon in the header

One way to customize your application is to introduce a custom icon in the header:

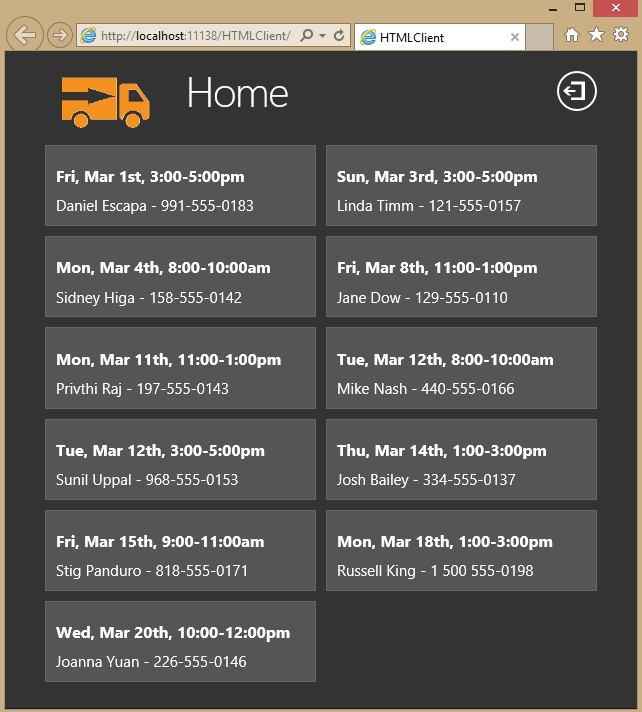
1. In the **Solution Explorer**, switch to **File View**. Under the **HTMLClient** node, expand the **Content** and **Images** nodes. You will see a default **user-logo.png** and **user-splash-screen.png** file.



1. Delete **user-logo.png** (or rename it).
2. Open the shortcut menu for the **Images** folder and choose **Add,** **Existing Item**. Navigate to the **Sample Resources** folder (provided as part of the sample) and choose **user-logo.png**. **Note:** You may choose any other PNG file to serve as a logo, but you will need to re-name it to **user-logo.png** once you add it to the project.
3. Similarly, you can also customize the image on the splash screen by replacing the **user-splash-screen.png**.
4. Hit **F5** and run the application! Please note that you may need to clear your browser cache to see the new logo image.   
     
   

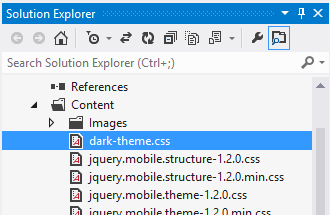
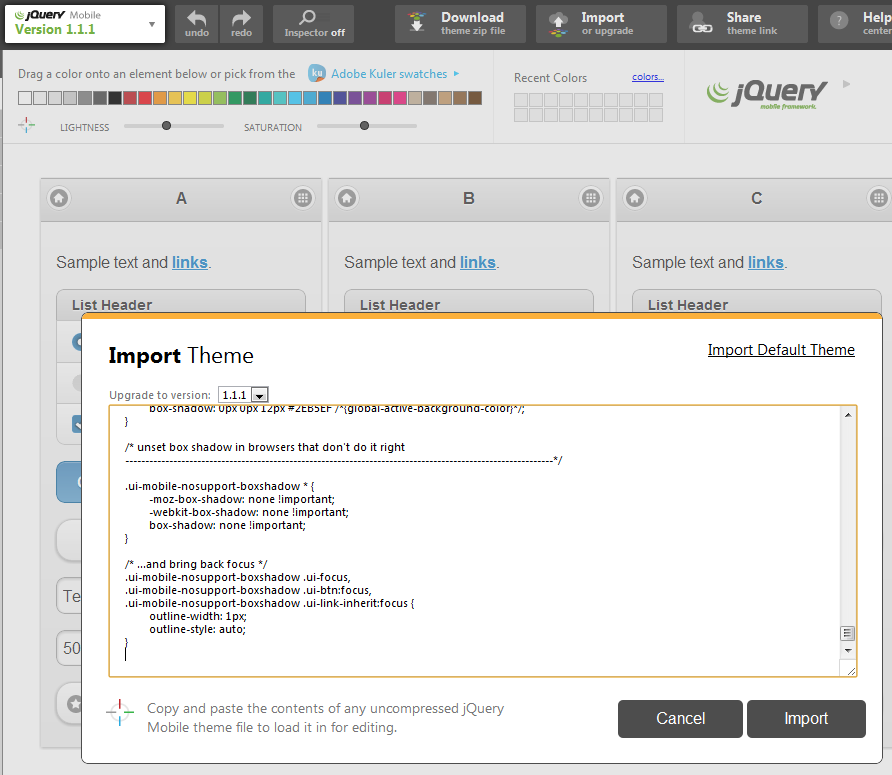
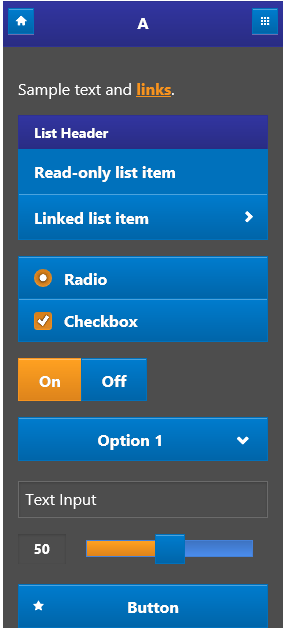
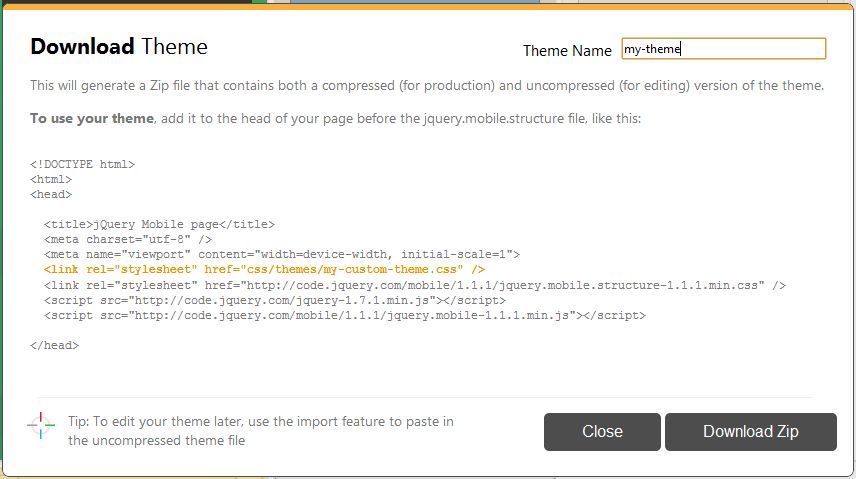
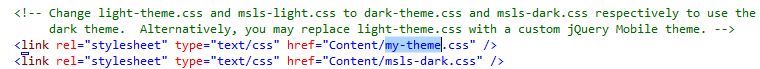
## Switch between “Light” and “Dark” themes

Out of the box, LightSwitch comes with two themes: a **light** (default) and a **dark** theme. To switch between the themes:

1. In the **Solution Explorer**, switch to **File View**. Under the **HTMLClient** node, open **default.htm**.
2. Around line 20, you will find the following:  
   <link rel="stylesheet" type="text/css" href="Content/light-theme.css" />  
   <link rel="stylesheet" type="text/css" href="Content/msls-light.css" />
3. To switch between the light and dark themes, simply substitute **light** or **dark** into the **href** portion of the 2 lines. To use the dark theme:   
   <link rel="stylesheet" type="text/css" href="Content/dark-theme.css" />  
   <link rel="stylesheet" type="text/css" href="Content/msls-dark.css" />
4. F5 and run the application!  
   

## Integrate ThemeRoller themes

The appearance of a LightSwitch Mobile app can be customized via **ThemeRoller for JQueryMobile**. You can start building your own theme, or you can customize on top of the light/dark theme provided out of the box. To customize the existing light theme:

1. Go to the **ThemeRoller** website at <http://jquerymobile.com/themeroller/>
2. Click **Import** button on the top of **ThemeRoller** page.
3. Go back to **Visual Studio IDE**, open **dark-theme.css** under **Content**.  
   
4. Copy and paste the CSS content into the **Import Theme** dialog in **ThemeRoller**. Click **Import**.   
   
5. Use **ThemeRoller** to customize **Global** and **Swatch A**. For example:  
   
6. Click **Download** button on the top.
7. Name your theme **my-theme** and click **Download Zip**.  
   
8. Extract the zip file on your desktop.
9. From **Solution Explorer**, open the shortcut menu for the **Content** folder and choose **Add, Existing Item…**
10. Navigate to the downloaded file and add **my-theme.css**.
11. Open **default.htm** and replace the **light-theme.css** with **my-theme.css**  
    
12. Press **F5** and run the application.  
    