Survey Application Tutorial:

Developing a SharePoint Application Using LightSwitch

Microsoft LightSwitch for Visual Studio 2012 Update 2

LightSwitch now supports the ability to create SharePoint applications that can be easily installed to and launched from a SharePoint site. LightSwitch SharePoint applications automatically handle identity flow between SharePoint and the LightSwitch application and provide a code experience for interacting with SharePoint assets. This tutorial will show you how to use LightSwitch to build a SharePoint application with an HTML client that runs on a variety of mobile devices.

Contents

[Survey Application Overview 2](#_Toc350179941)

[Step 1: Create an HTML LightSwitch Application 2](#_Toc350179942)

[Step 2: Add Support for Viewing Survey Data 3](#_Toc350179943)

[2.1 Add Survey Entity 3](#_Toc350179944)

[2.2 Add Home Screen for Displaying Surveys 5](#_Toc350179945)

[Step 3: Enable SharePoint 7](#_Toc350179946)

[Step 4: Publish to SharePoint 12](#_Toc350179947)

[Step 5: Add Support for Creating and Editing Survey Data 16](#_Toc350179948)

[5.1 Attach to Customer and Product SharePoint Lists 16](#_Toc350179949)

[5.2 Relate Survey to Customer and Product Entities 19](#_Toc350179950)

[5.3 Add Screen for Creating and Editing Surveys 22](#_Toc350179951)

[5.4 Add Defaulting/Filtering Logic for Sales Rep Name 25](#_Toc350179952)

[5.5 Update the Customers Last Survey Date 28](#_Toc350179953)

[5.5 Update the splash screen 30](#_Toc350179954)

[Congratulations! 31](#_Toc350179955)

[Step 6: Add Support for Uploading and Displaying Photos 32](#_Toc350179956)

[6.1 Add Photo Entity 32](#_Toc350179957)

[6.2 Relate Survey to Photo Entity 32](#_Toc350179958)

[6.3 Add Photos Picture Library 33](#_Toc350179959)

[6.4 Add a WebAPI Controller to Interact with Photos Picture Library 35](#_Toc350179960)

[6.5 Authenticate the Client to Interact with the Photos Picture Library 40](#_Toc350179961)

[6.6 Add Custom Control for Uploading Photos 40](#_Toc350179962)

[6.7 Add Screen Support for Displaying Photos 43](#_Toc350179963)

[6.8 Add Support for Deleting Photos 48](#_Toc350179964)

[Step 7: Republish to SharePoint and Verify Runtime Behavior 50](#_Toc350179965)

# Survey Application Overview

*Contoso Foods* is both a food manufacturer and distributor that sells a variety of products to grocery stores nationwide. *Contoso Foods’* Sales Representatives play an important role in maintaining relationships with partner stores by frequently visiting each location to deliver products and conduct quality surveys. Quality surveys are completed for every product to measure the presence that the product has within the store. Typical survey data that is collected by Sales Representatives includes:

* Cleanliness of the product display (ranging from “very poor” to “excellent”)
* Quality of the product display lighting (also ranging from “very poor” to “excellent”)
* Product location within an aisle (middle of aisle, end of aisle, or aisle end-cap)
* Product shelf height position (top shelf, eye-level shelf, or bottom shelf)

In addition, as part of completing surveys, photos are taken of the product display to support the overall assessment.

On a weekly basis, Sales Representatives visit store locations to take product surveys. Currently, survey data is captured using a clipboard and pen, but this method is slow and increases the likelihood of transcription errors. Also, this method makes it difficult to take and attach photos to surveys. To address these problems, the sales team has decided to create a Survey Application that Sales Representatives can access from their tablet devices to easily collect survey data and attach photos that have been taken using their device. Specifically, the Survey Application will be an Office 365 SharePoint application created using Visual Studio LightSwitch. Key reasons for this approach are:

* The sales team recently switched to Office 365 for internal email and collaboration, so Sales Representatives are already used to signing into the team’s SharePoint Online site to view customer contact information, marketing material, and customer invoices. Based on this, the team’s SharePoint site is the logical place to host and launch the Survey Application.
* SharePoint Online offers easy access and management of images. SharePoint’s Picture Library automatically creates thumbnail and web optimized versions of images which improves performance when displaying photos within the application.

This tutorial will walk you through the steps for developing the Survey Application that *Contoso Foods’* Sales Representatives will use for completing survey assessments.

# Step 1: Create an HTML LightSwitch Application

Begin by launching Visual Studio 2012 as an administrator. When the **New Project** dialog is displayed, select one of the following project templates to create a LightSwitch HTML client application:

* **LightSwitch HTML Application (Visual Basic)**
* **LightSwitch HTML Application (Visual C#)**

Set the project and solution name to **SurveyApplication** and select a location to save the solution. Click the **OK** button to create and open the application within Visual Studio.

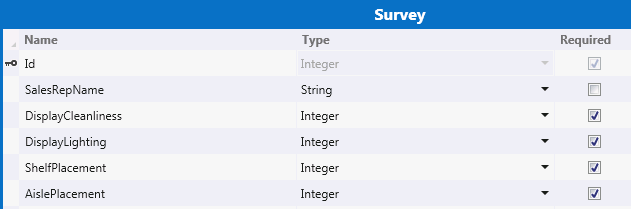
At this point your LightSwitch application is a standard web application – there is nothing specific for SharePoint added to the application just yet. In a later step, you’ll be shown how to enable your application to use SharePoint so that you can easily host your application on a SharePoint site, use SharePoint assets within your application, and more.

# Step 2: Add Support for Viewing Survey Data

A core aspect of the application will be to display survey data. To add this functionality, let’s begin by adding a **Survey** entity in LightSwitch’s intrinsic database that will be used for storing survey data.

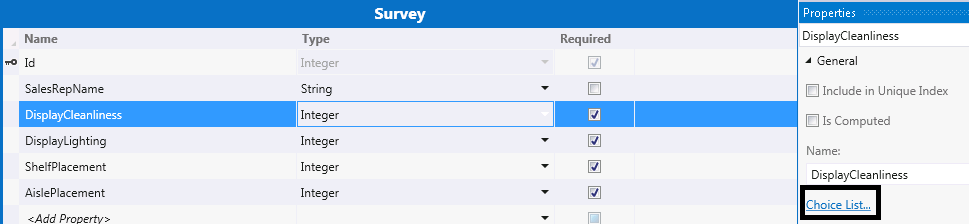
## 2.1 Add Survey Entity

1. In the **Solution Explorer**, right-click the **Server** node and select **Add Table**.
2. When the entity designer opens, rename the entity to **Survey**.
3. Use the entity designer to add the properties as shown in the below screenshot:

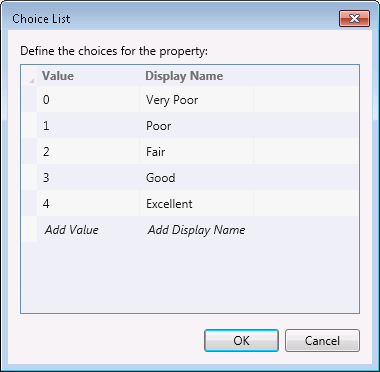


Ensure that the SalesRepName property’s Required checkbox is unchecked; all of the other properties should be required.

1. Select the DisplayCleanliness property on the Survey entity. In the Properties Window, click the Choice List… hyperlink.



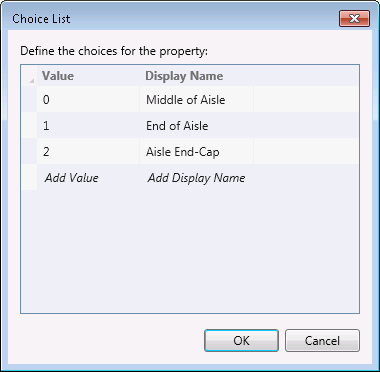
In the **Choice List** dialog, add values according to the below screenshot and click **OK** to save changes and close the dialog:



1. Open the **Choice** **List** dialog for the **DisplayLighting** property and add the same values as listed above for the **DisplayCleanliness** property.
2. Open the **Choice List** dialog for the **ShelfPlacement** property and add values according to the below screenshot.



1. Open the **Choice List** dialog for the **AislePlacement** property and add values according to the below screenshot.

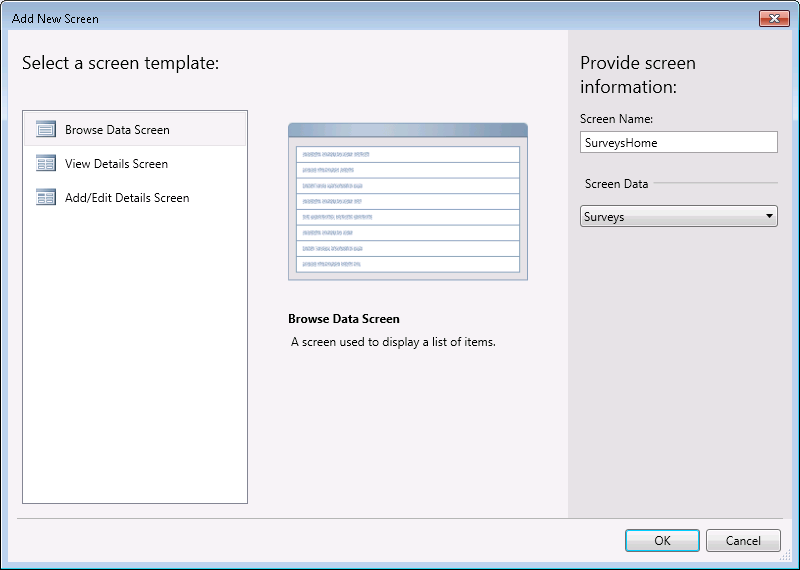


## 2.2 Add Home Screen for Displaying Surveys

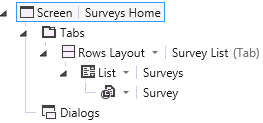
The home screen is the first screen that is displayed when the application is launched. Since survey data is central to the Survey Application, let’s create a home screen that is bound to the **Surveys** table.

1. In the **Solution Explorer**, right-click the **Client** node and select **Add Screen…**
2. In the **Add New Screen** dialog:
   1. Select the **Browse Data Screen** template.
   2. Set the screen name to **SurveysHome**.
   3. Select **Surveys** from the **Screen Data** drop-down list.
   4. Click **OK** to save changes and close the dialog.

Note that this screen will be set as the home screen by default.

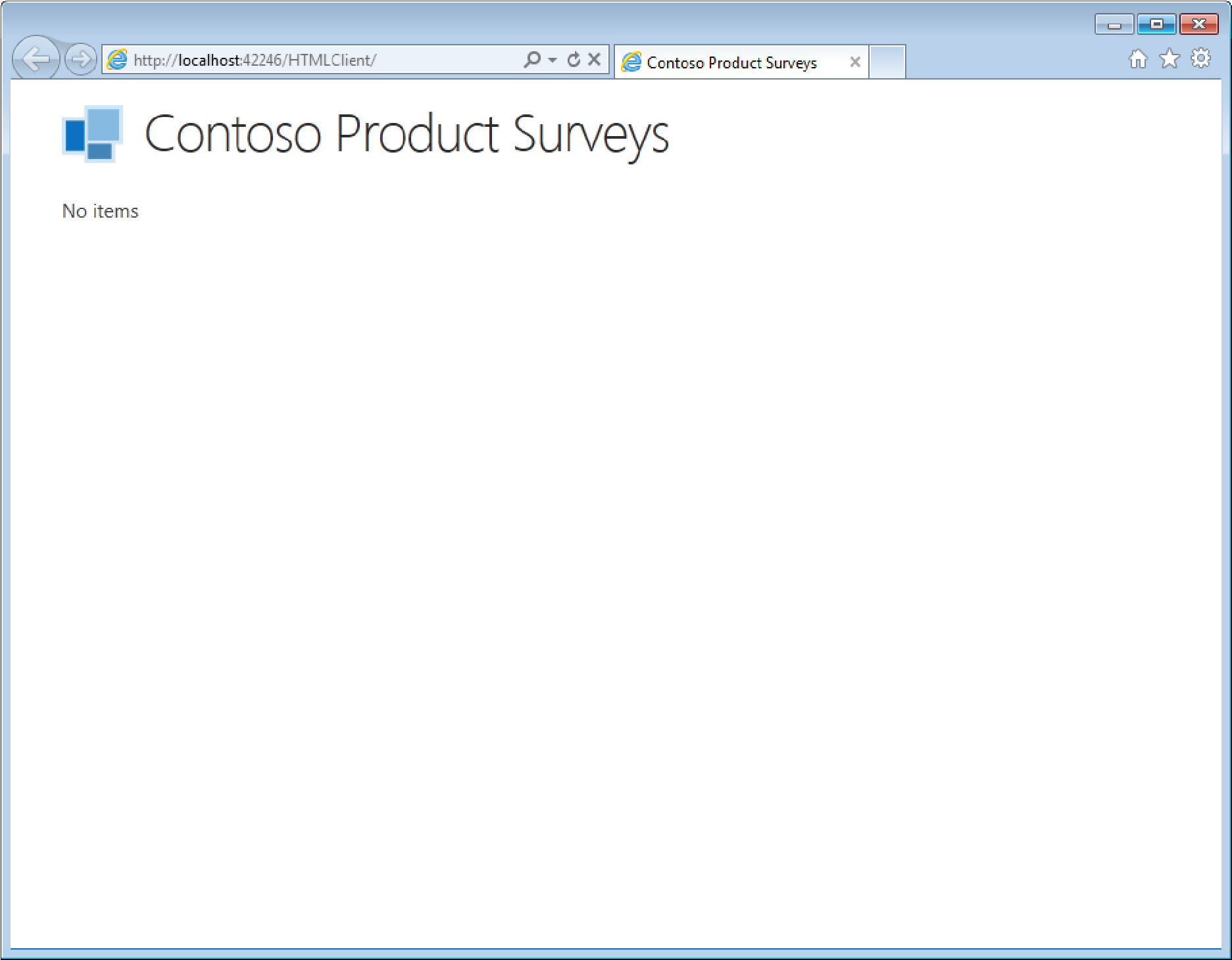


1. In the screen designer, select the **Surveys Home** node and set the **Display Name** property to **Contoso Product Surveys** in the **Properties Window**.



F5 Check Point: Build and Run the Application

Let’s verify the functionality that we’ve added so far by pushing the F5 key to build and run the application. Since this is a standard web application, you will see the web browser launch and the **SurveysHome** screen displayed:



There are no survey items displayed by the screen since we haven’t entered any survey data into the intrinsic database yet. In later steps, we’ll add a screen that will be used for both creating and editing survey data.

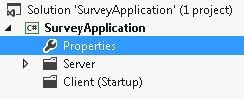
Stop the application from running by closing the web browser window that was launched so that we can continue with developing the rest of the application.

# Step 3: Enable SharePoint

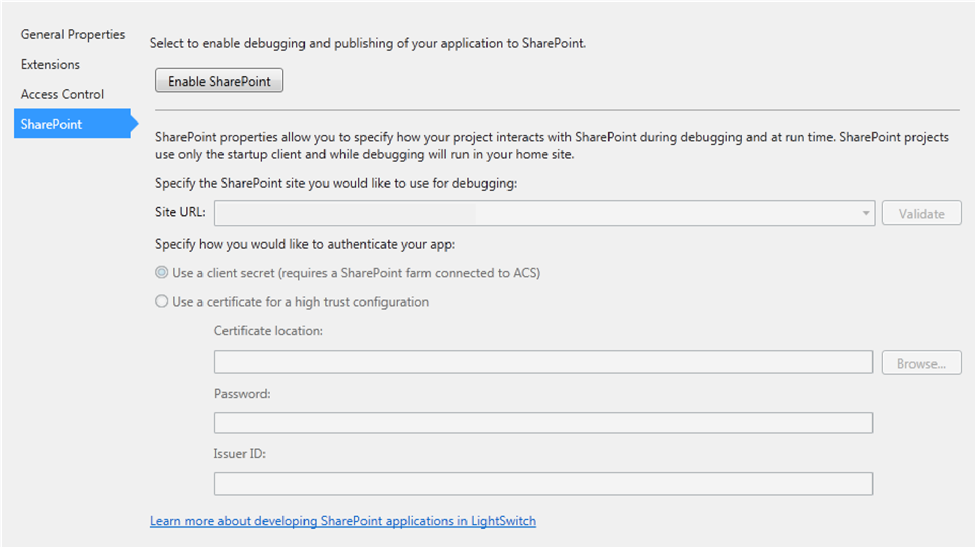
Up until now, we’ve been working with a standard HTML Client project. Since our goal is to create an application that can be easily accessed and launched from a SharePoint Online site, we need to modify the project settings to enable SharePoint. Hosting the application on SharePoint will provide users with an intuitive experience since they are already accustomed to interacting with SharePoint to view documents and related content. In addition, users will now benefit from the rich data and functionality provided by our LightSwitch application. With SharePoint enabled, our application has the ability to integrate with a variety of SharePoint features, including search, workflow, etc. Furthermore, our application can access various SharePoint assets ranging from picture libraries to calendars.

IMPORTANT: *To enable SharePoint, you must first* [*sign up for an Office 365 Developer Site*](http://msdn.microsoft.com/en-us/library/office/apps/fp179924(v=office.15))*. This site will be used for developing and testing the remainder of the Survey Application.*

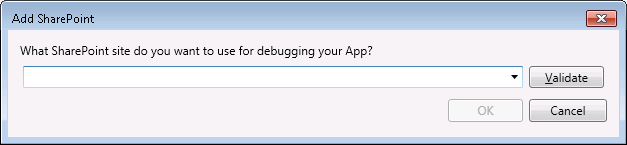
1. In the **Solution Explorer**, double-click the root project’s **Properties** node to open the project settings.



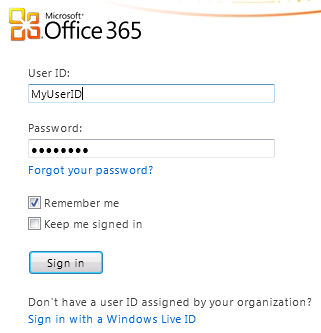
1. Select the **SharePoint** tab.



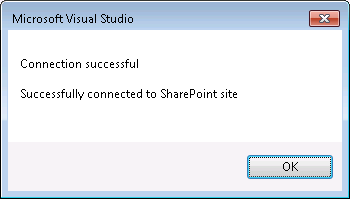
1. Click the **Enable SharePoint…** button. When the **Add SharePoint** dialog opens, specify the URL for the Office 365 Developer Site that will be used for hosting your application. This URL is based on the user ID and domain that you provided when you [signed up](http://msdn.microsoft.com/en-us/library/office/apps/fp179924(v=office.15)) for the developer site. The format of this URL is as follows: *userid@yourdomain.onmicrosoft.com.*



1. Click the **Validate** button to ensure that connection succeeds. When prompted, enter the SharePoint credentials for the developer site and click **Sign in** (note that if you’ve already signed into the site, you will not be prompted).

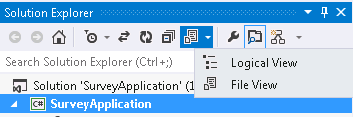


Upon successful connection, you will see the following dialog. Click **OK** to continue.

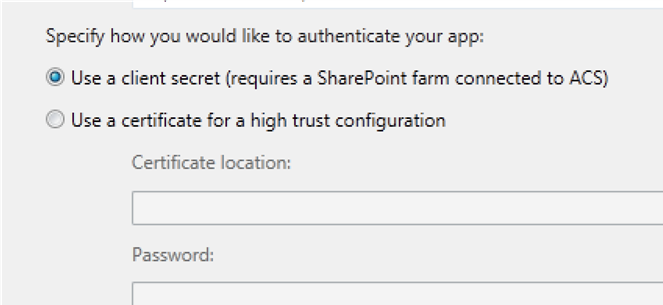


1. Click the **OK** button to close the **Add SharePoint** dialog. As a result of enabling SharePoint, your LightSwitch project now has references to several SharePoint assemblies and has a nested SharePoint web application project. Also, our application now has access to [SharePoint’s client object model (CSOM)](http://msdn.microsoft.com/en-us/library/fp179912(v=office.15).aspx#ClientAPIs) to interact with SharePoint features and assets.

You can view the SharePoint web application project by switching to **File View** in the **Solution Explorer**. Use the **Toggle View** button in the **Solution Explorer’s** toolbar to navigate between the **File View** and **Logical View**.



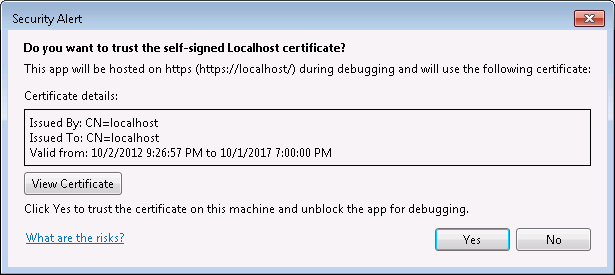
1. Choose the authentication type for your SharePoint app. Since O365 uses ACS (Access Control Service), we will be using a client secret.

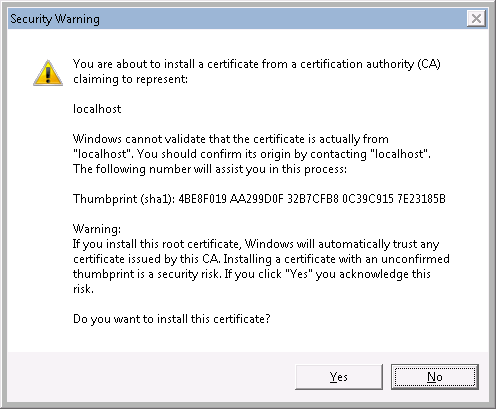


F5 Check Point: Build and Run the Application

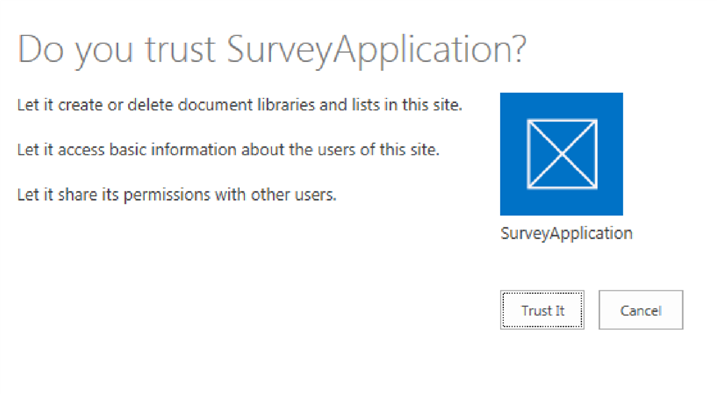
You can now build and run your application to see it hosted on the Office 365 Developer Site.

1. Push the F5 key to build and run the application. The first time that you run a SharePoint application on your computer, you will be prompted to install a **Localhost certificate** since SharePoint enabled LightSwitch applications always use SSL (Secure Sockets Layer). Accepting this certificate allows you to run the application on your computer without receiving a security warning each time you launch it. Because the certificate only applies to Localhost, there is no security threat to your system. Click **Yes** to install the certificate. If you receive a **Security Warning**, also click **Yes**.

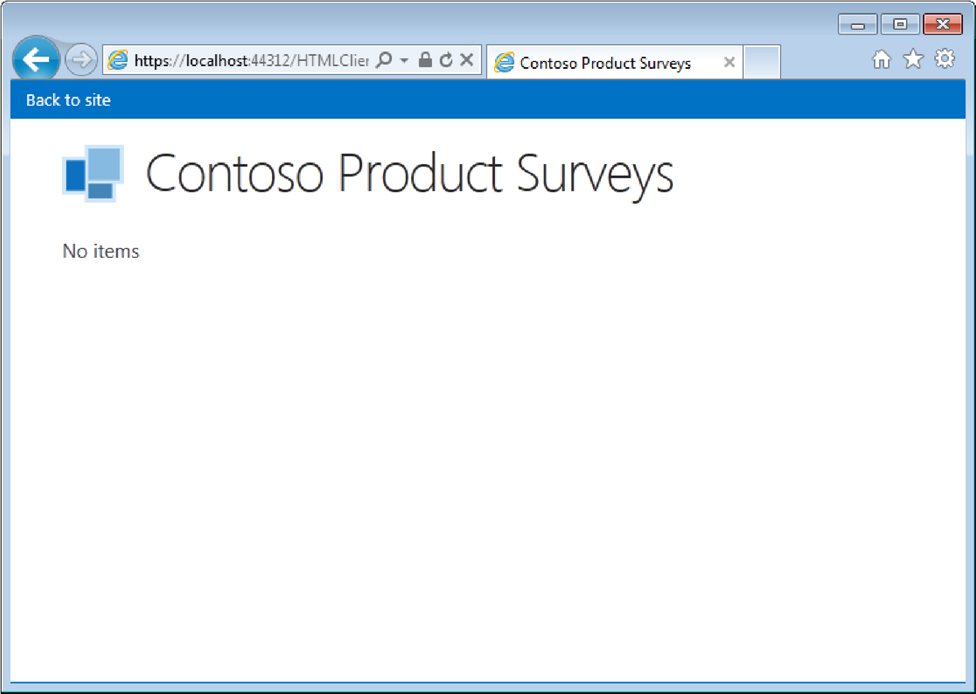




1. When the web browser launches, SharePoint will prompt you to select whether you trust the application before running it. The application requires access to basic site and user information in order to authenticate the SharePoint user. Click **Trust It** so that the SharePoint Survey Application launches.



1. As a result, the Survey Application will launch with the home screen displayed. Notice that the application now has a [SharePoint chrome bar](http://msdn.microsoft.com/en-us/library/fp179916(v=office.15).aspx) displayed at the top of the screen that allows the user to navigate back to the SharePoint site.



Behind the scenes, LightSwitch uses [OAuth for authentication and authorization with SharePoint](http://msdn.microsoft.com/en-us/library/office/apps/fp142382(v=office.15).aspx). The OAuth flow begins with the user signing into the SharePoint site so that they are properly authenticated. When the LightSwitch application is launched from SharePoint, a context token is passed from SharePoint to the application – this context token contains information about the current authenticated user. As a result, when user information is retrieved from within the application, it is for the same user that is signed into SharePoint. Similarly, when the application uses CSOM for interacting with SharePoint features and assets, the same context token is used to communicate back into SharePoint. Specific examples showing the importance of OAuth flow will be provided later in the tutorial, but for now, the important thing to remember is that LightSwitch handles all of this automatically for you.

1. Stop the application from running by closing the web browser window.

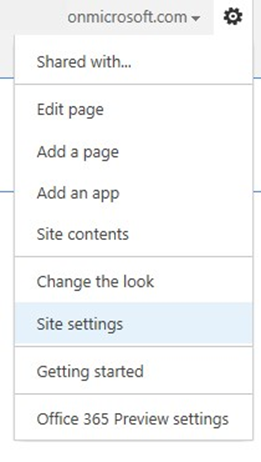
# Step 4: Add Support for Creating and Editing Survey Data

Now that we’ve seen how to incorporate SharePoint into our application, let’s get back to adding functionality to the application itself so that it supports the ability to create and edit survey data.

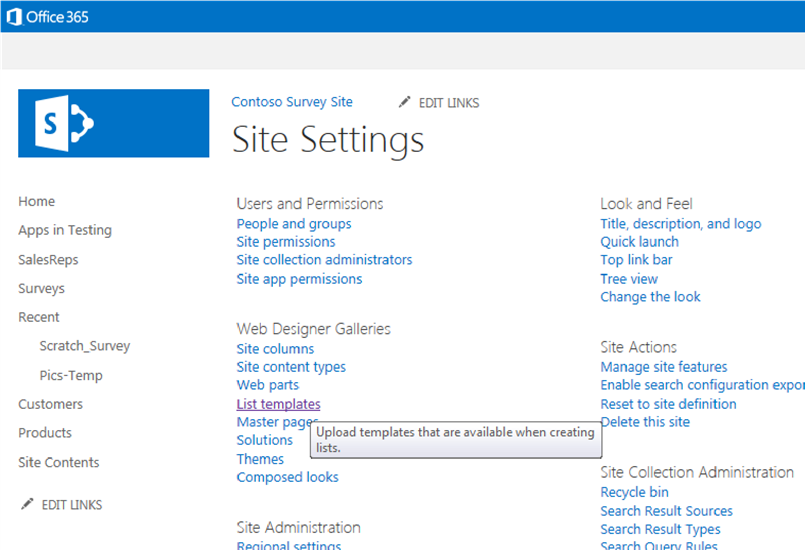
## 4.1 Attach to Customer and Product SharePoint Lists

First, let’s make a few additions to the underlying data model. When a new survey is created, we need the ability to relate the survey to a specific customer and product. A real-world application would integrate with existing customer and product data that could be stored in the cloud, in an on-premise SQL server, in SharePoint, etc. For the purposes of this tutorial, we will attach to some existing SharePoint lists that contain **Customer** and **Product** data. You can simply import the template files that are included with the tutorial. This will create the entities you can attach to and they will already be populated with data.

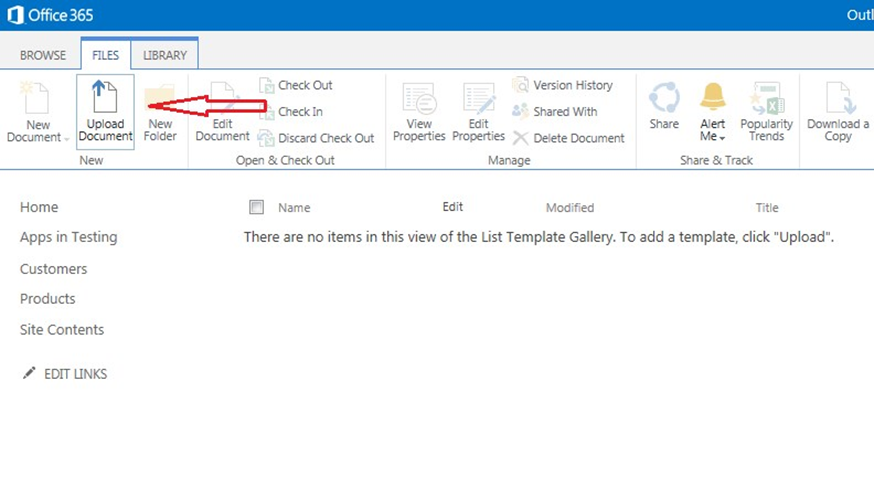
1. Navigate to your O365 developer site.
2. Choose **Site Settings** from the settings menu



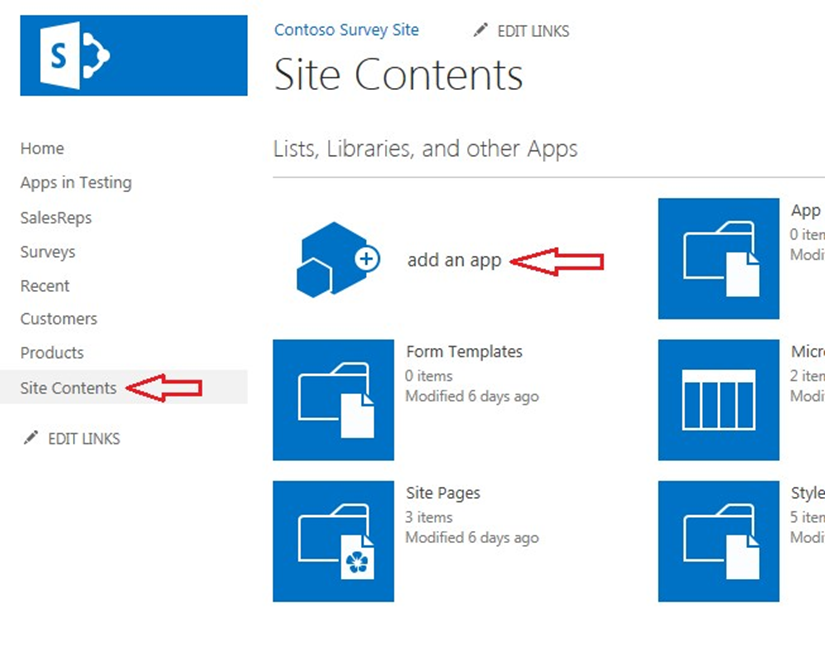
1. From **Site Settings** navigate to the **List templates** gallery



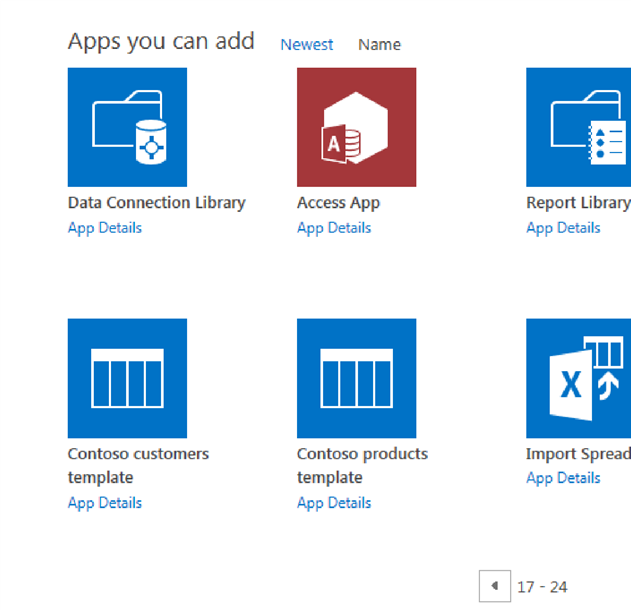
1. From the **Files** tab choose **Upload Document.** When prompted for a file choose the **contoso\_customers\_template.stp** file included with the tutorial. Then click **Save** on the next dialog to confirm the upload.



1. Repeat the upload steps for the contoso\_**products\_template.stp** file.
2. After saving the template we need to create lists from each template. These templates also contain the sample data we’ll use for the surveys. Choose **Site Contents** from the quick links on your SharePoint site. Next, choose **add an app**.



Look for the customer and product templates we just added.

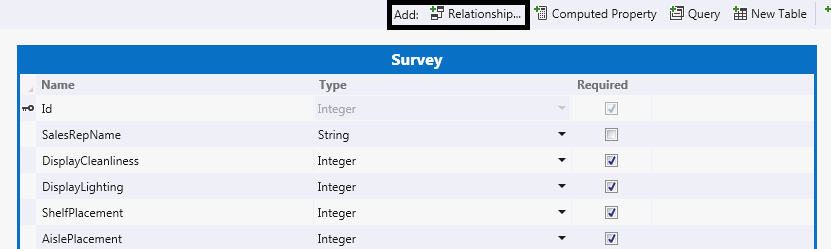


1. Click each to install the list and data onto your site. When prompted, use the name **Customers** for the customer list and **Products** for the product list. You now have 2 lists with data that you can attach to from LightSwitch.
2. Back to **Visual Studio**, In the **Solution Explorer**, right-click the **Server** node and select **Add Datasource.**
3. When the wizard opens choose **SharePoint** and click **Next.**
4. For the SharePoint site address enter the URL of your O365 dev site. Let LightSwitch automatically choose the identity to use. Using this option, LightSwitch will use the credentials of the user running the application to access the list data. Click **Next** and enter your username and password for the SharePoint site if prompted.
5. When prompted for the entities to use in your application, select the **Customers** and **Products** entities. The **UserInformation** list will be selected for you automatically. Click **Finish** to continue working on the application.
6. Build the application and verify there are no compile errors.

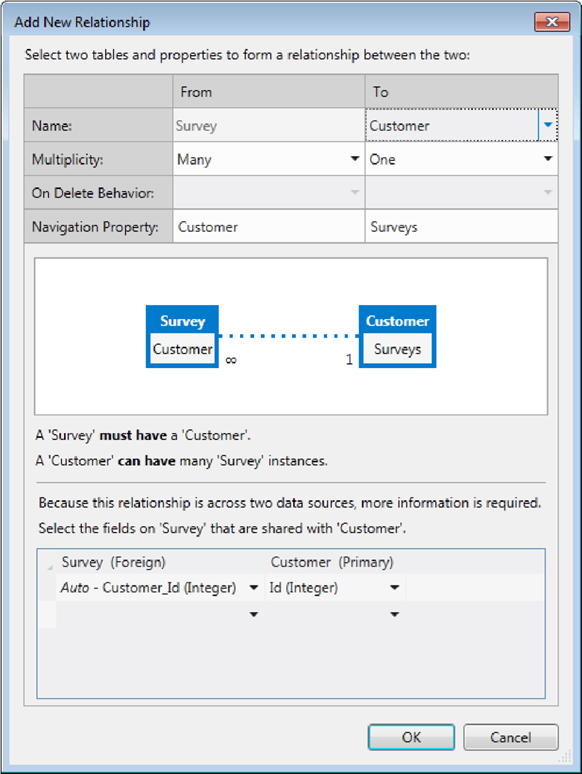
## 4.2 Relate Survey to Customer and Product Entities

Next, we need to add relationships from the **Survey** entity to the **Customer** and **Product** entities.

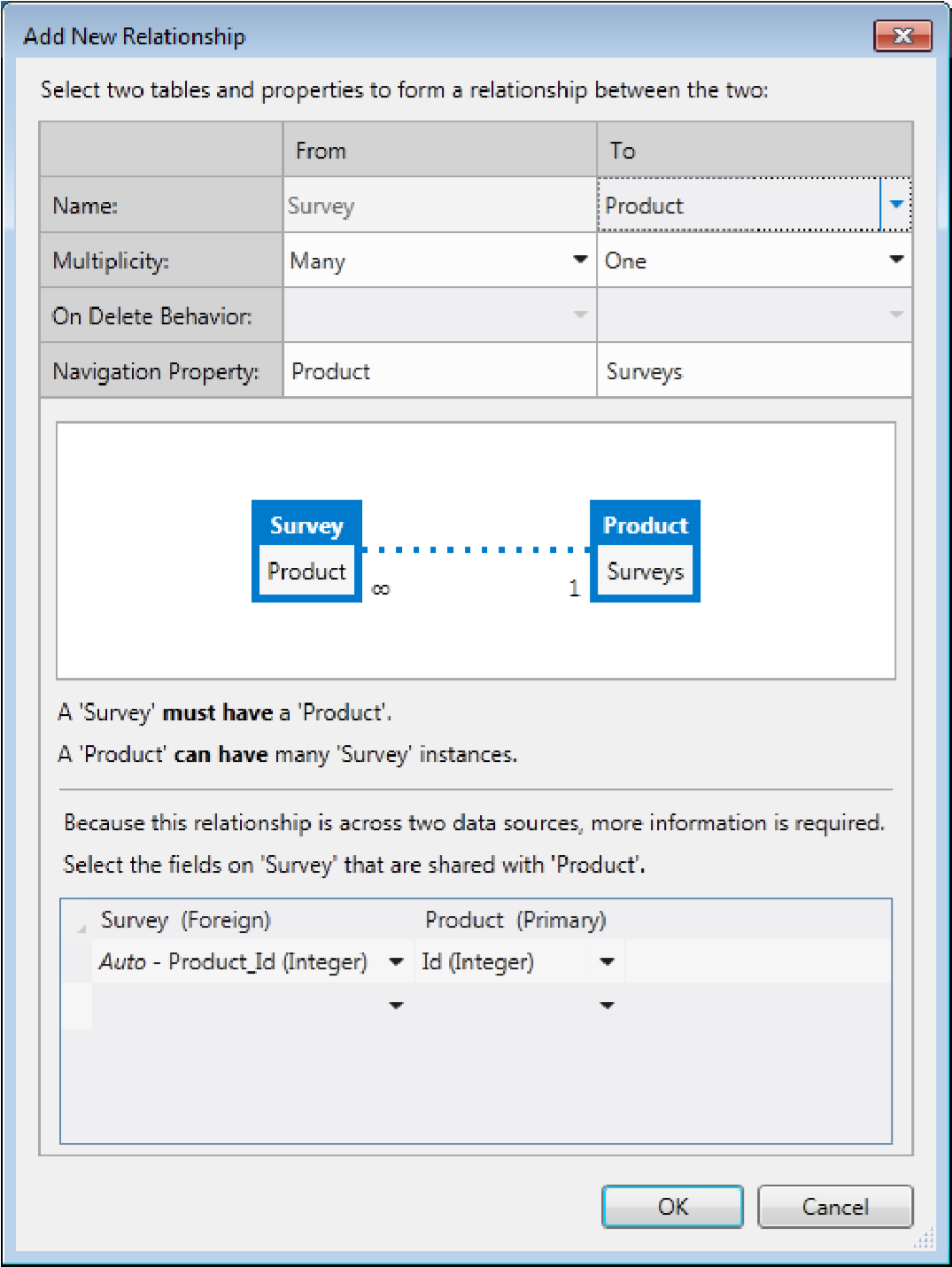
1. Add a many-to-one relationship between **Survey** and **Customer**.
   1. In the **Solution Explorer**, switch to **Logical View**.
   2. In the **Solution Explorer**, under the **Server ApplicationData** node, double-click the **Surveys** table to open the entity designer.
   3. Select the **Survey** entity in the entity designer and click **Add: Relationship**.



* 1. In the **Add New Relationship** dialog, set the **To** property to **Customer** and leave all of the other properties set to their default value. Click **OK** to save changes and close the dialog:

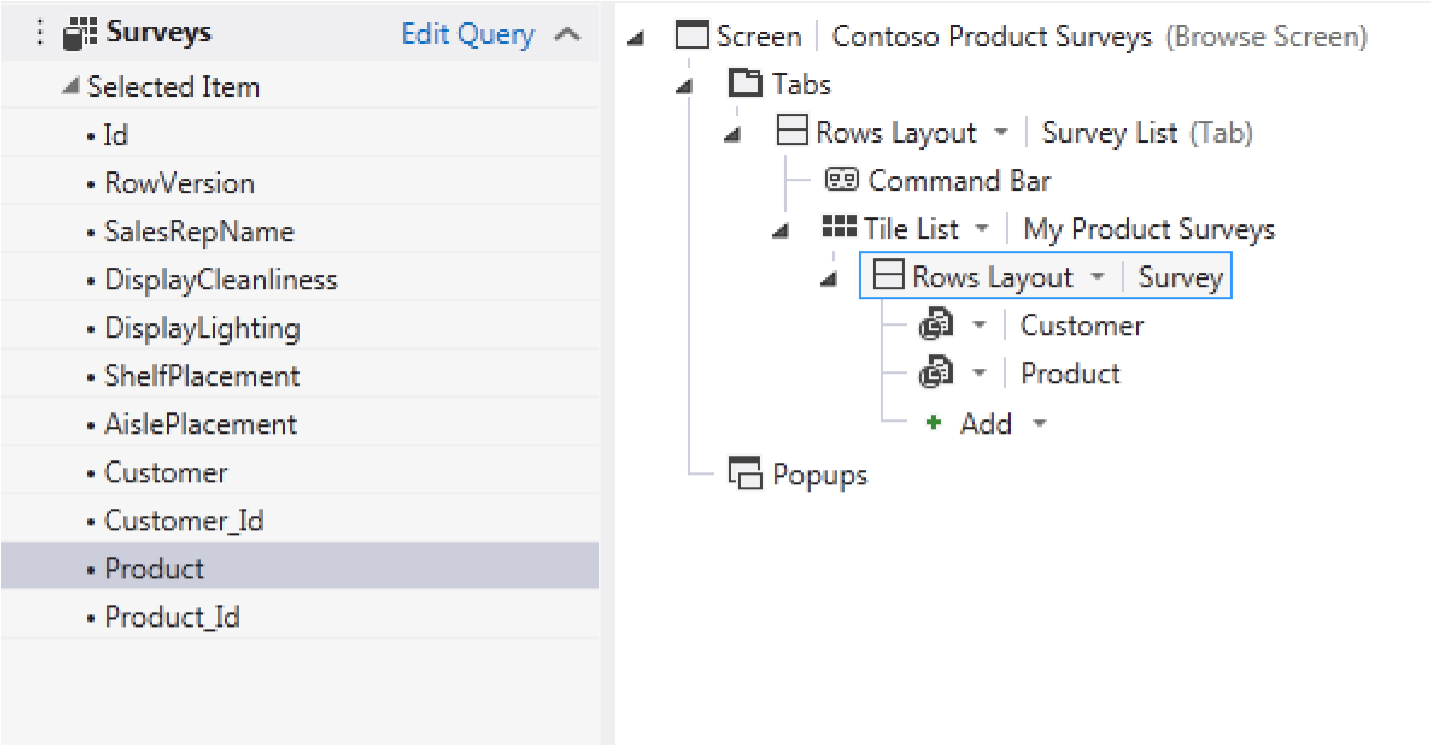


1. Add a many-to-one relationship between **Survey** and **Product**.
   1. Click **Add: Relationship**.
   2. In the **Add New Relationship** dialog, set the **To** property to **Product** and leave all of the other properties set to their default value. Click **OK** to save changes and close the dialog.



Finally, let’s improve the information that is displayed by the home screen’s survey list so that it includes customer and product information.

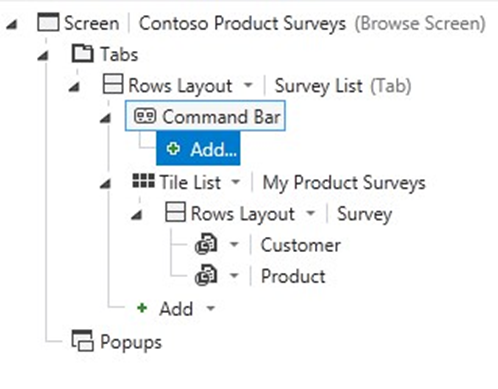
1. In the **Solution Explorer** under the **Client** node, double-click the **SurveysHome** screen to open the screen designer.
2. Select the **Surveys** list node and change the list type to **Tile List.**
3. Delete the properties that are automatically added to **Rows Layout** and drag-and-drop the **Customer** and **Product** properties so that they are displayed.



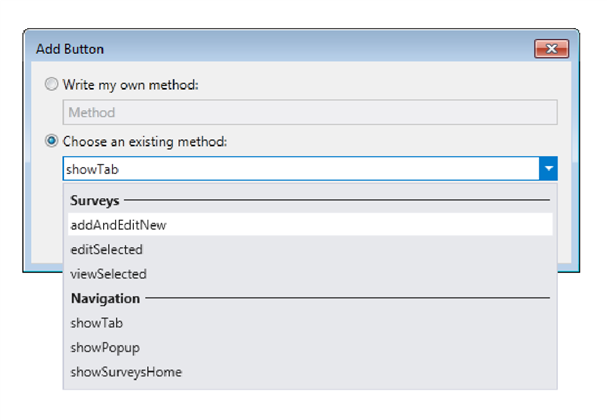
## 4.3 Add Screen for Creating and Editing Surveys

In this step, we will add a screen to add new surveys. The screen will open when the user clicks on a button to create a new survey or taps on an existing survey.

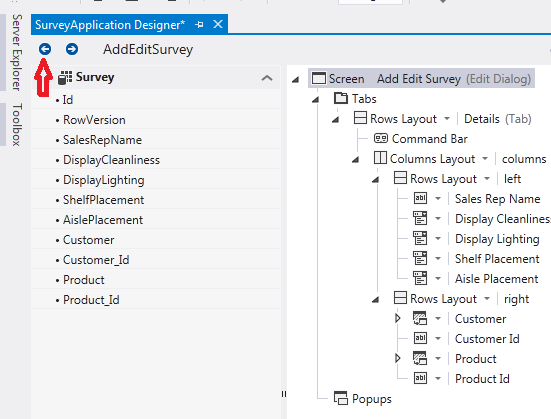
1. Select the **Command Bar** node under the **Survey List** rows layout node and click **Add…**



1. In the **Add Button** dialog:
   1. Choose the **existing method** option and select **addAndEditNew**
   2. Set Navigate To (**New Screen…**)
   3. Click **OK** to save changes and close the dialog.

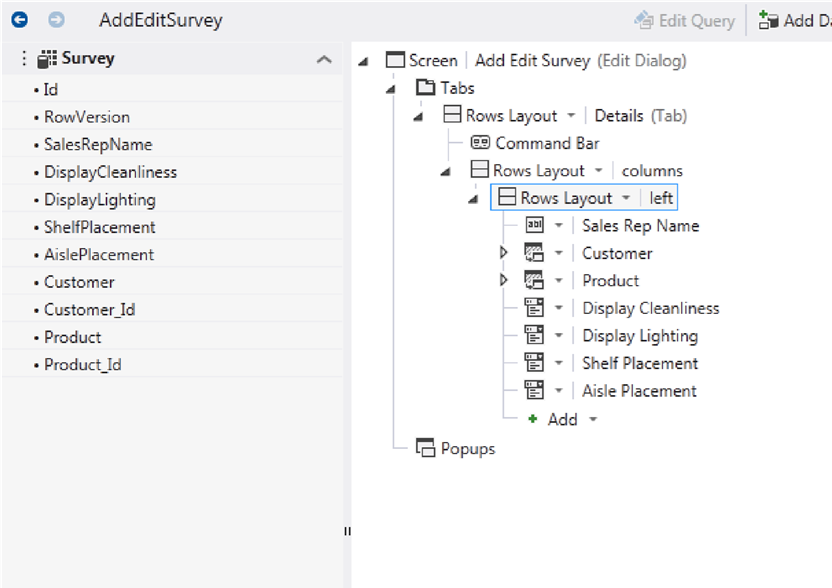


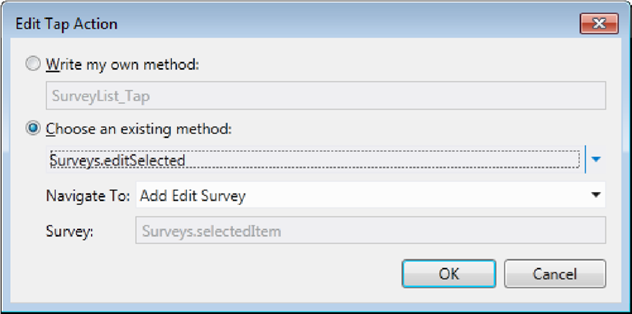
* 1. On the **Add New Screen** dialog, accept the defaults and click **OK.**
  2. Click the back button in the screen designer to go back to the **SurveysHome** screen



* 1. Select the **Add Survey** command button node.
  2. In the **Properties** window for the command button, set the **Icon** type to **Add**.

1. Click the forward button in the Screen Designer to open the designer for the **AddEditSurvey** screen and then remove the **Customer\_ID** and **Product\_ID** properties from the screen.
2. Next, drag the **Customer** and **Product** property to the left rows layout and delete the right rows layout.



1. In Solution Explorer, double click on the **SurveysHome** screen to open the screen designer. Select the **Tile List** node for the Survey List. In the **Properties Window**, click the **Item Tap** action hyperlink under the **Actions** group.
   1. Choose the **existing method** option and select **editSelected** under **Surveys**.
   2. Set navigation to **Add Edit Surveys**.
   3. Click **OK** to save changes and close the dialog.  
      

F5 Check Point: Build and Run the Application

Let’s verify the new screen and the navigation that we just added.

1. Push the F5 key to build and run the application.
2. When the web browser launches, click the **Add** button that exists on the home screen. This will navigate to the **AddEditSurveys** screen and automatically create a new **Survey** entity instance.



1. Open the **Customer** list and notice that customer data from the SharePoint list is displayed.
2. Similarly, open the **Product** list and notice that product data from the SharePoint list is displayed.
3. Finally, expand each of the remaining lists and verify that you see the choice list values that we created earlier in the tutorial.
4. Stop the application from running by closing the web browser window.

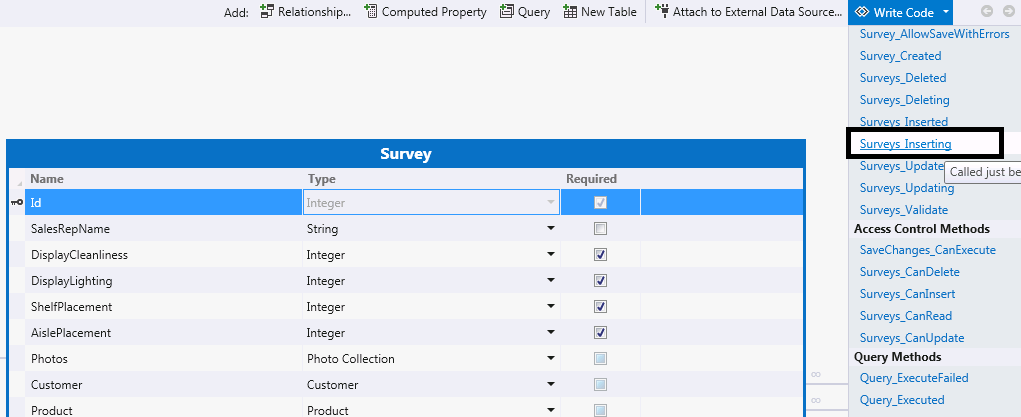
Before creating and saving survey data, there are still a few changes that we can make to improve the application’s user experience:

* Automatically default the **Sales Rep Name** to the name of the current authenticated user.
* Filter the surveys so that the current authenticated user only sees surveys that they have created.

## 4.4 Add Defaulting/Filtering Logic for Sales Rep Name

When the user is creating a new **Survey**, let’s automatically default the **Sales Rep Name** property to the current authenticated user so that the user doesn’t have to enter this data. In addition, we’ll add filter logic so that the current authenticated user only sees surveys that they have created.

1. In the **Solution Explorer**, double-click the **Surveys** table to open the entity designer.
2. In the entity designer, select the **Survey** entity and ensure the **Server** perspective is selected.
3. Open the **Write Code** drop-down list and choose to override the **Surveys\_Inserting** method. This method is executed within the server’s save pipeline each time that a new **Survey** entity is in process of being inserted.



1. When the code editor opens for the **ApplicationDataService.[cs|vb]** file, insert the following code into the **Surveys\_Inserting** method so that the **SalesRepName** property is defaulted to the current authenticated user:

**VB:**

Private Sub Surveys\_Inserting(entity As Survey)

entity.SalesRepName = Application.User.Name

End Sub

**C#:**

partial void Surveys\_Inserting(Survey entity)

{

entity.SalesRepName = this.Application.User.Name;

}

Remember that this is made possible by the OAuth flow that LightSwitch handles automatically for us. When the user information is retrieved from within our application, it is for the same user that is already signed into SharePoint. In fact, if you were to inspect the **User.Name** property value while your application is running, you’d notice that the value consists of the SharePoint log-in name along with additional information about the user’s identity. The format of this property value is similar to the following: *i:0#.f|membership| userid@yourdomain.onmicrosoft.com.*

1. Next, let’s add filter logic so that the current authenticated user only sees surveys that they have created. To do this, reopen the entity designer and select the **Survey** entity. Open the **Write Code** drop-down list and choose to override the **Surveys\_Filter** method. This method is executed on the server each time the **Survey** entity is queried.
2. When the code editor opens for the **ApplicationDataService.[cs|vb]** file, insert the following code into the filter method so that the when the **Survey** entity is queried, the surveys are filtered based on the current authenticated user.

**VB:**

Private Sub Surveys\_Filter(ByRef filter As System.Linq.Expressions.Expression(Of System.Func(Of Survey, Boolean)))

filter = Function(e) e.SalesRepName = Application.User.Name

End Sub

**C#:**

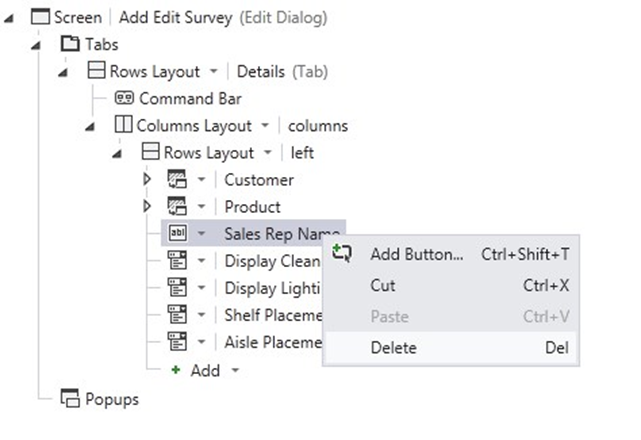
partial void Surveys\_Filter(ref Expression<Func<Surveys, bool>> filter)

{

filter = f => f.SalesRepName == this.Application.User.Name;

}

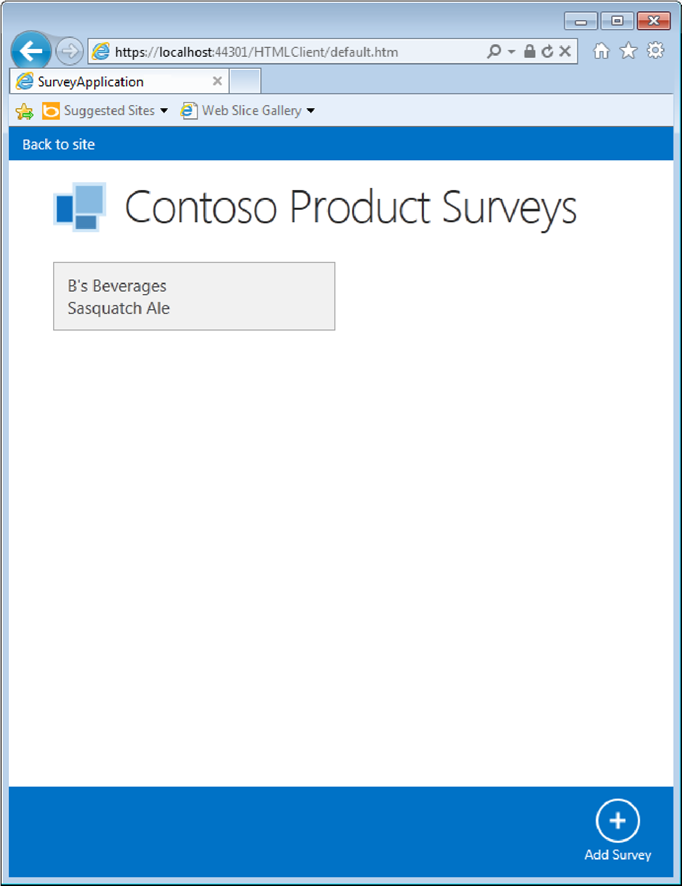
1. Now that we’ve added a filter so that the current authenticated user only sees their survey information, the **Sales Rep Name** property that is displayed on the **AddEditSurveys** screen no longer provides value. Based on this, let’s remove the **Sales Rep Name** property from the screen. Specifically, in the **Solution Explorer**, double-click the **AddEditSurveys** screen to open the screen designer and delete the screen’s **Sales Rep Name** property.



F5 Check Point: Build and Run the Application

Let’s verify the defaulting and filtering logic that we added.

1. Push the F5 key to build and run the application.
2. When the web browser launches, click the **Add** button that exists on the home screen.
3. Complete the survey assessment by setting all of the properties. Click the save button.
4. Navigate back to the home screen and verify the survey that you just created is displayed in the list of surveys.



1. Stop the application from running by closing the web browser window.

## 4.5 Update the Customers Last Survey Date

Finally, let’s update the **Customers** SharePoint list to reflect the last date this customer site was surveyed for any product.

1. In the **Solution Explorer**, double-click the **Surveys** node under the **Server** node.
2. In the entity designer, select the **Survey** entity and ensure the **Server** perspective is selected.
3. Open the **Write Code** drop-down list and choose to override the **Surveys\_Inserted** method. This method is executed within the server’s save pipeline each time that a new **Survey** entity is in inserted.
4. When the code editor opens for the **ApplicationDataService.[cs|vb]** file, insert the following code into the **Surveys\_Inserted** method so that the **LastSurveyDate** property is defaulted to today for the customer being surveyed. Be sure to replace the highlighted text with the name of your datasource that contains the attached SharePoint lists.

**VB:**

Private Sub Surveys\_Inserted(entity As Survey)

Dim c As Customer = Me.DataWorkspace.Contoso\_Survey\_SiteData.Customers.Where(

Function(f) f.CompanyName = entity.Customer.CompanyName).FirstOrDefault()

If (Not c Is Nothing) Then

c.LastSurveyDate = Date.Today

Me.DataWorkspace.Contoso\_Survey\_SiteData.SaveChanges()

End If

End Sub

**C#:**

partial void Surveys\_Inserted(Survey entity)

{

Customer c = this.DataWorkspace.Contoso\_Survey\_SiteData.Customers.Where(f =>

f.CompanyName == entity.Customer.CompanyName).FirstOrDefault();

if (c != null)

{

c.LastSurveyDate = DateTime.Today;

this.DataWorkspace.Contoso\_Survey\_SiteData.SaveChanges();

}

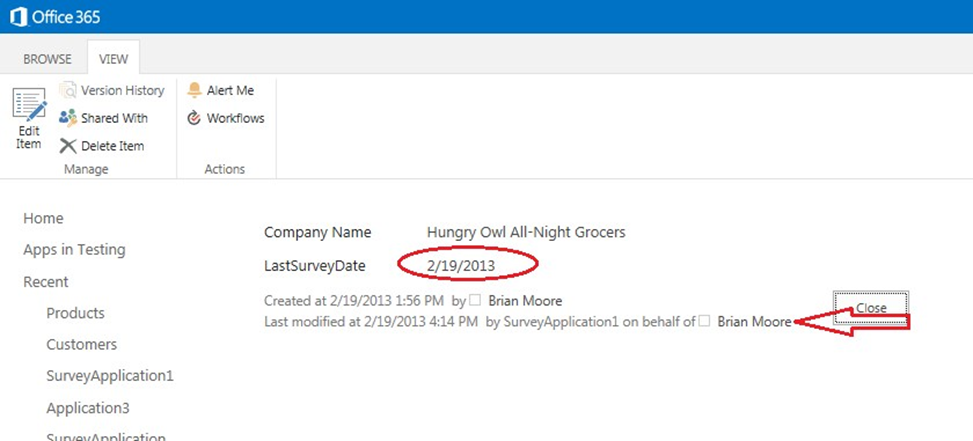
}

F5 Check Point: Build and Run the Application

Let’s verify the logic that we added.

1. Push the F5 key to build and run the application.
2. When the web browser launches, click the **Add** button that exists on the home screen.
3. Complete the survey assessment by setting all of the properties. Click the save button.

Navigate to your SharePoint site and view the Customers list. Notice that the **LastSurveyDate** has been updated for the customer you just surveyed. Additionally, the item in the SharePoint list shows that it has been changed by the user running the app.



## 4.6 Update the splash screen

1. In the **Solution Explorer**, switch to **File View.**
2. In the **Solution Explorer**, select the **Client** project and double click on **default.htm**.
3. Change the HTML below to make the splash screen reflect the name of the application. To change the name displayed on the opening screen, change the text for the DIV element to **Contoso Product Surveys.**

<body>

<div id="msls-id-app-loading" class="ui-body-a msls-layout-ignore">

<div class="msls-app-loading-img"></div>

<span class="ui-icon ui-icon-loading"></span>

<div class="ui-bottom-load">

<div>Contoso Product Surveys</div>

</div>

</div>

Additionally, change the TITLE element, to display the correct name in the title bar or tab of the browser.

<head>

<meta charset="utf-8" />

<meta http-equiv="X-UA-Compatible" content="IE=10" />

<meta name="HandheldFriendly" content="true" />

<meta name="viewport" content="width=device-width, initial-scale=1, minimum-scale=1, maximumscale=1, user-scalable=no" />

<meta name="apple-mobile-web-app-capable" content="yes" />

<title>Contoso Product Surveys</title>

F5 Check Point: Build and Run the Application

Verify that the splash screen and the title now show the name of your application

# Congratulations!

We now have a complete survey application. At this point you can publish your application to your SharePoint site and add and edit new surveys. To really round out the application and leverage more SharePoint functionality, we can incorporate pictures into the survey results. The rest of the walk through will guide you through those steps.

Note, if you plan to publish the application on a new SharePoint site, be sure to install the Customers and Products SharePoint lists on the new site by repeating the instructions in [5.1 Attach to Customer and Product SharePoint Lists](#_5.1_Attach_to), and when publishing, change the connect string for the SharePoint site, to point to your new site.

# Step 5: Add Support for Uploading and Displaying Photos

In this phase of building the Survey Application, we will add support for uploading photos to a SharePoint Picture Library. When a photo is uploaded to a Picture Library, SharePoint automatically creates two additional image sizes based on the original uploaded version:

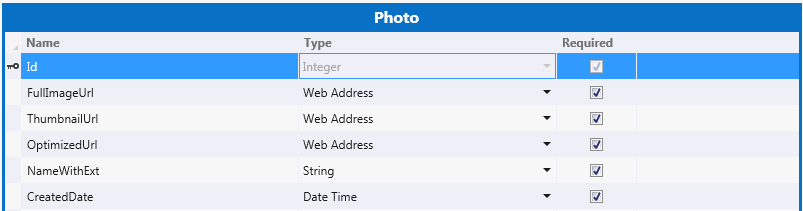
1. Thumbnail image
2. Web optimized image

While LightSwitch intrinsically supports storing images in its SQL database, a SharePoint Picture Library is a more fitting mechanism for persisting and managing images than a database – it allows users to browse images easily and the application to display the thumbnail and web optimized versions of the image (which will perform better than displaying the full sized image).

## 5.1 Add Photo Entity

First, we need to add a **Photo** entity to the intrinsic database. As already mentioned, the photos themselves will be stored in a SharePoint Picture Library. The purpose of the **Photo** entity is simply to reference the image URLs so that the photos can be displayed within the Survey Application.

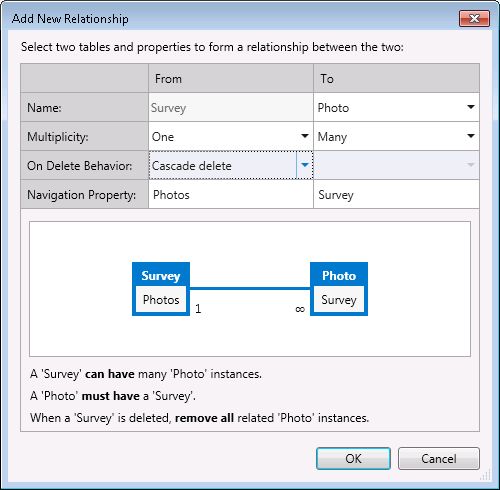
1. In the **Solution Explorer**, right-click the **Server** node and select **Add Table**.
2. When the entity designer opens, rename the entity to **Photo**.
3. Use the entity designer to add the properties as shown in the below screenshot:



## 5.2 Relate Survey to Photo Entity

Next, we need to add a one-to-many relationship from the Survey entity to the Photo entity.

1. Open the Survey entity in the entity designer and click A**dd: Relationship**.
2. In the **Add New Relationship** dialog, set the relationship properties according to the below screenshot; click **OK** to save changes and close the dialog:

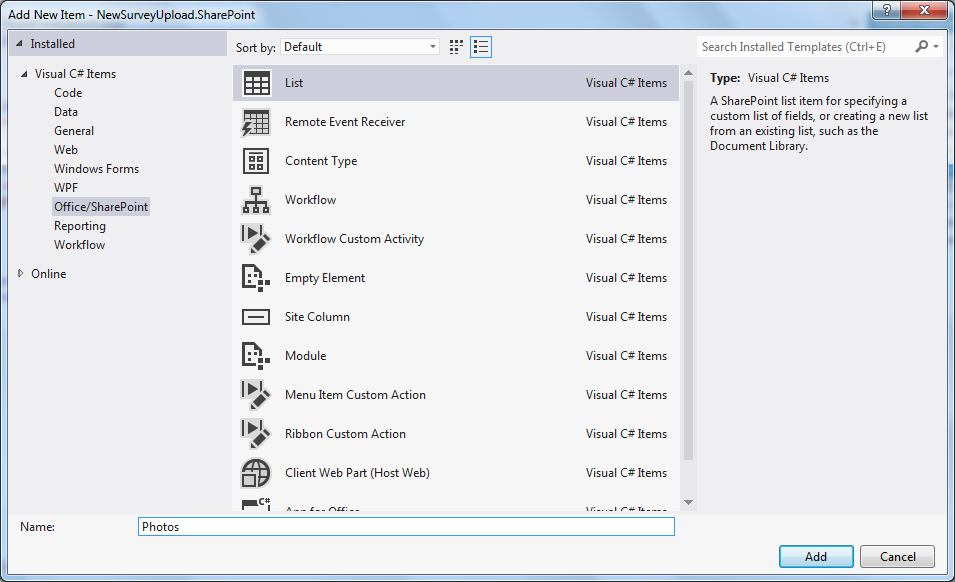


## 5.3 Add Photos Picture Library

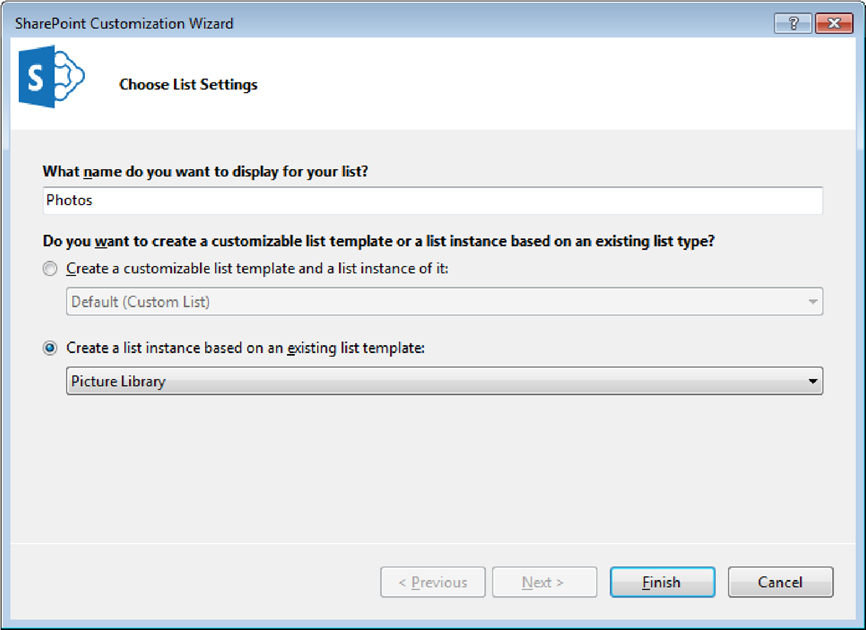
SharePoint applications may include a variety of assets such as lists, modules, content types, property bags, etc. that are installed as part of the application. Assets deployed as part the application reside in the [*app web*](http://msdn.microsoft.com/en-us/library/fp179925%28v=office.15%29.aspx)*.* Assets in the app web will be installed and removed in conjunction with the corresponding application.

Creating assets, such as custom lists, in the app web is a useful technique when an application requires SharePoint capabilities in an isolated fashion. For example, in the Survey Application, we will use a Picture Library for storing photos. It is important to note, however, that any content stored in the app web will be removed when the application is uninstalled.

1. In the **Solution Explorer**, switch to **File View**.
2. Right-click the **SharePoint** project and select **Add** followed by **New Item…**
3. Under **Office/SharePoint**, select the **List** and rename the list to **Photos**. Click **Add**.



1. When the **SharePoint Customization Wizard** opens, select the **Create a non-customizable list …** option and choose **Picture Library** as the list type. Click **Finish** to close the wizard.



1. Lastly, add **PhotoListHelper.[cs|vb]** to the server project:
2. Right-click the **Server** project and select **Add** followed by **Existing Item…**
3. Browse to the location of the **PhotoListHelper.[cs|vb]** source file and click **OK** to add the file to the project.

This simple utility uses the CSOM API to interact with the **Photos** Picture Library to upload and delete images. Notice that the utility requires a **Microsoft.SharePoint.Client.ClientContext** instance be passed in. The **ClientContext** serves as the entry point for interacting with SharePoint assets – in this case, a Picture Library located in the app web context. The app web **ClientContext** is easily retrieved using LightSwitch’s server API. This will be discussed in further detail in the next section.

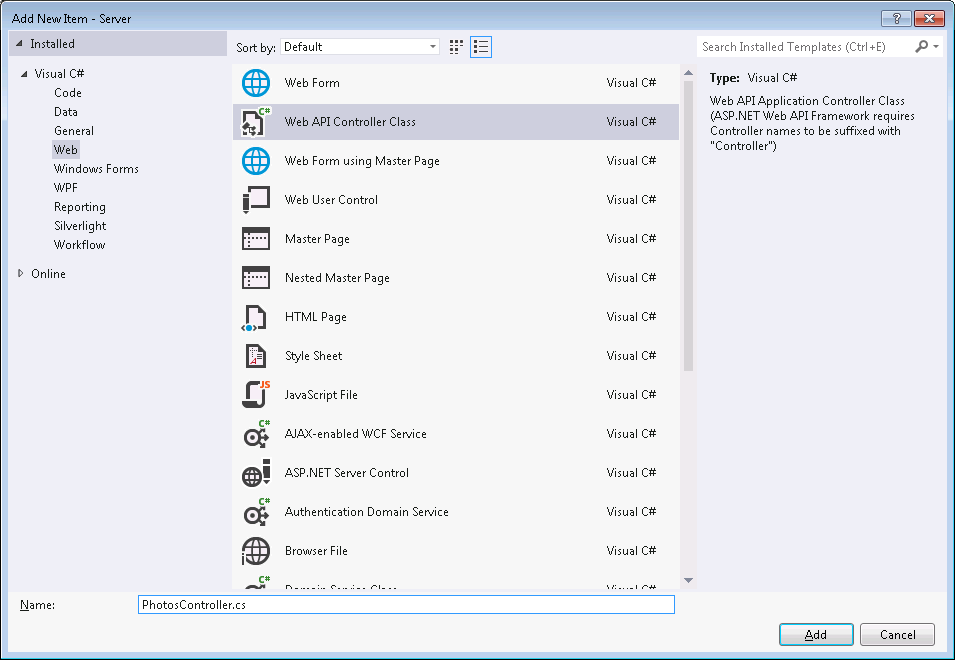
1. Build the application and verify there are no compile errors.

## 5.4 Add a WebAPI Controller to Interact with Photos Picture Library

Although LightSwitch exposes the CSOM API on the server, it is not exposed on the client. As a result, we need another mechanism to upload photos from the client to the **Photos** Picture Library. Specifically, we’ll use [WebAPI](http://www.asp.net/web-api).

1. Remain in **File View** for this part of the tutorial.
2. In the **Solution Explorer**, select the **Server** project and select **Add** followed by **New Item…**
3. Under Web, select **Web API Controller Class** and rename the class to **PhotosController**. Click **Add** to close the dialog.

IMPORTANT: WebAPI uses convention-based routing, so please make sure to use the name **PhotosController**; otherwise, the AJAX calls we’ll make from the client may not be routed to the controller and files won’t be uploaded.



1. Add the following using/imports to the controller:

**VB:**

Imports LightSwitchApplication

Imports Microsoft.SharePoint.Client

Imports System.IO

Imports System.Net.Http

Imports System.Net.Http.Headers

Imports System.Text

Imports System.Threading.Tasks

**C#:**

using Microsoft.SharePoint.Client;

using System.IO;

using System.Net.Http.Headers;

using System.Text;

using System.Threading.Tasks;

1. Replace the placeholder controller methods with the methods listed below.

IMPORTANT: *Make sure that you delete the placeholder methods that are added to the controller by default, otherwise the code will fail to compile.*

Notice that this code retrieves the **SharePoint** host object using LightSwitch’s server API (refer to the highlighted line). The **SharePoint** host object provides access to several useful properties and methods used for interacting with the SharePoint site. In particular, the **GetAppWebClientContext** method provides an entry point for interacting with assets belonging to the app web. Remember that LightSwitch automatically handles the OAuth flow so that when the app web’s **ClientContext** is retrieved, the proper context token is used for communicating back into SharePoint.

**VB:**

Private \_appWebContext As ClientContext

Private ReadOnly Property AppWebContext() As ClientContext

Get

If \_appWebContext Is Nothing Then

Using serverContext = LightSwitchApplication.ServerApplicationContext.CreateContext()

\_appWebContext = serverContext.Application.SharePoint.GetAppWebClientContext()

End Using

End If

Return \_appWebContext

End Get

End Property

Public Function PostFormData() As Task(Of HttpResponseMessage)

If Not Request.Content.IsMimeMultipartContent() Then

Throw New HttpResponseException(HttpStatusCode.UnsupportedMediaType)

End If

Dim memStream = New MultipartMemoryStreamProvider()

Dim spCtx = AppWebContext

Dim myTask = Request.Content.ReadAsMultipartAsync(memStream).ContinueWith(

Function(t)

If t.IsFaulted OrElse t.IsCanceled Then

Request.CreateErrorResponse(HttpStatusCode.InternalServerError, t.Exception)

End If

Dim fileList = New StringBuilder()

For Each contentItem In memStream.Contents

Dim sourceFileName = Path.GetFileName(contentItem.Headers.ContentDisposition.FileName.Replace("""", ""))

Try

Dim fileReadTask = contentItem.ReadAsByteArrayAsync().ContinueWith(

Function(frt)

Dim photoUrl = PhotoListHelper.AddPhoto(frt.Result, sourceFileName, spCtx)

Return photoUrl

End Function

)

fileReadTask.Wait()

fileList.AppendLine(fileReadTask.Result)

Catch ex As Exception

fileList.AppendLine("FAILED::" + ex.Message)

End Try

Next

Return Request.CreateResponse(HttpStatusCode.Created, fileList.ToString(), New MediaTypeHeaderValue("application/xml"))

End Function

)

Return myTask

End Function

Public Function DeletePhoto(url As String) As Task(Of HttpResponseMessage)

Dim spCtx = AppWebContext

Dim deleteTask = New Task(Of HttpResponseMessage)(

Function()

Try

PhotoListHelper.DeletePhoto(url, spCtx)

Return Request.CreateResponse(HttpStatusCode.OK)

Catch ex As Exception

Return Request.CreateErrorResponse(HttpStatusCode.NotModified, ex)

End Try

End Function

)

deleteTask.Start()

Return deleteTask

End Function

**C#:**

private ClientContext appWebContext;

private ClientContext AppWebContext

{

get

{

if (appWebContext == null)

{

using (var serverContext = LightSwitchApplication.ServerApplicationContext.CreateContext())

{

appWebContext = serverContext.Application.SharePoint.GetAppWebClientContext();

}

}

return appWebContext;

}

}

public Task<HttpResponseMessage> PostFormData()

{

if (!Request.Content.IsMimeMultipartContent())

{

throw new HttpResponseException(HttpStatusCode.UnsupportedMediaType);

}

// Since we're uploading the image to Sharepoint directly, we'll just read the Http content into memory

var memoryStream = new MultipartMemoryStreamProvider();

// We need to get the appweb context before forking the new task; otherwise the LightSwitch runtime will

// throw because there is no HttpContext available on the calling thread.

var sharepointContext = AppWebContext;

var task = Request.Content.ReadAsMultipartAsync(memoryStream).

ContinueWith<HttpResponseMessage>(t =>

{

if (t.IsFaulted || t.IsCanceled)

{

Request.CreateErrorResponse(HttpStatusCode.InternalServerError, t.Exception);

}

// There can be multiple files in the POST, so we'll upload each attachment and attach its new

// URL (in the SP Picture Library).

StringBuilder fileList = new StringBuilder();

foreach (var contentItem in memoryStream.Contents)

{

var sourceFileName = Path.GetFileName(contentItem.Headers.ContentDisposition.FileName.Replace("\"", ""));

try

{

// Read the contents of the file into memory and upload it to Sharepoint

var fileReadTask = contentItem.ReadAsByteArrayAsync().

ContinueWith<string>(frt =>

{

var photoURL = PhotoListHelper.AddPhoto(frt.Result, sourceFileName, sharepointContext);

return photoURL;

});

fileReadTask.Wait();

fileList.AppendLine(fileReadTask.Result);

}

catch (Exception ex)

{

fileList.AppendLine(@"FAILED::" + ex.Message);

}

}

return Request.CreateResponse(HttpStatusCode.Created, fileList.ToString(), new MediaTypeHeaderValue("application/xml")); ;

});

return task;

}

public Task<HttpResponseMessage> DeletePhoto(string url)

{

var sharepointContext = AppWebContext;

var deleteTask = new Task<HttpResponseMessage>(

() =>

{

try

{

PhotoListHelper.DeletePhoto(url, sharepointContext);

return Request.CreateResponse(HttpStatusCode.OK);

}

catch (Exception e)

{

return Request.CreateErrorResponse(HttpStatusCode.NotModified, e);

}

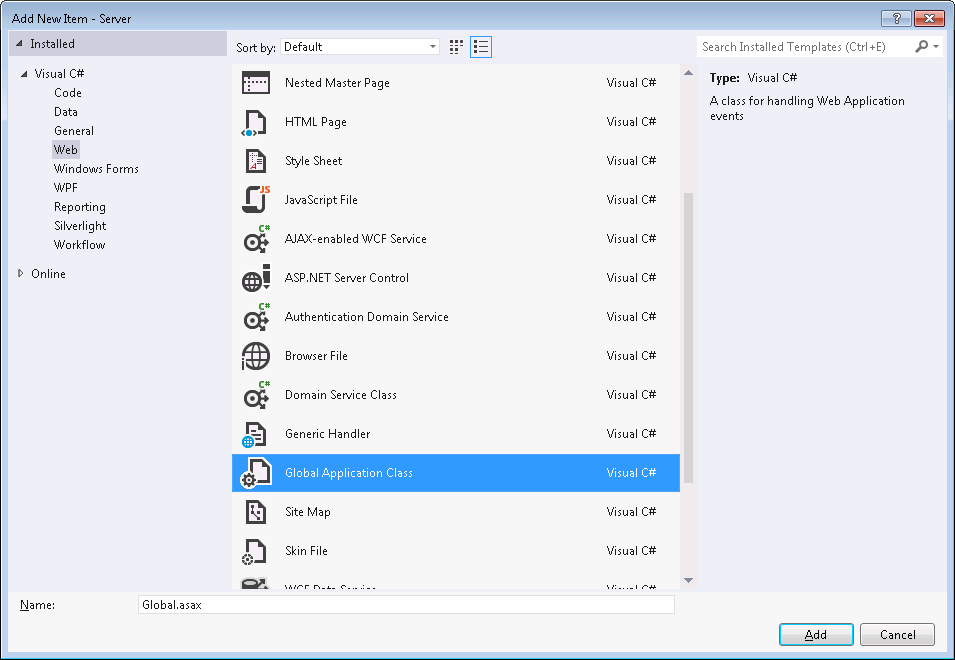
});

deleteTask.Start();

return deleteTask;

}

1. Now we just need to add an Http route to our controller so it can be invoked from the client. In the **Solution Explorer**, select the **Server** project and select **Add** followed by **New Item…** Under **Web**, select **Global Application Class** and leave the name set to **Global.asax**. Click **Add** to close the dialog.



1. Add the following using/imports to **Global.asax.[cs|vb]:**

**VB:**

Imports System.Web.Routing

Imports System.Web.Http

**C#:**

using System.Web.Routing;

using System.Web.Http;

1. Insert the following Http route into the **Application\_Start** method in **Global.asax.[cs|vb]**

**VB:**

Sub Application\_Start(ByVal sender As Object, ByVal e As EventArgs)

RouteTable.Routes.MapHttpRoute(

name := "DefaultApi",

routeTemplate := "api/{controller}/{id}",

defaults := New With { .id = System.Web.Http.RouteParameter.Optional }

)

End Sub

**C#:**

protected void Application\_Start(object sender, EventArgs e)

{

RouteTable.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = System.Web.Http.RouteParameter.Optional }

);

}

1. Build the application and verify there are no compile errors.

## 5.5 Authenticate the Client to Interact with the Photos Picture Library

The app web, which contains the **Photos** Picture Library, resides in an isolated domain. By default, the client is not authenticated to interact with the app web. As a result, we need to add a simple script to perform the app web authentication.

1. Remain in **File View** for this part of the tutorial.
2. In the **Solution Explorer**, select the **Client** project’s **~/Scripts/** folder and select **Add** followed by **Existing Item…**
3. Browse to the location of the **SharePointAuthHelper.js** file and click **OK** to add the file to the project.
4. Open the **Client** project’s **default.htm** file in the code editor. Add a reference in the file to **SharePointAuthHelper.js**.

<script type="text/javascript" src="Content/Resources/msls.default.prerequisite.resources.js"></script>

<script type="text/javascript" src="Scripts/msls-1.0.0.js"></script>

<script type="text/javascript" src="Scripts/sharepointauthhelper.js"></script>

<script type="text/javascript" src="Scripts/Generated/data.js"></script>

1. Build the application and verify there are no compile errors.

## 5.6 Add Custom Control for Uploading Photos

All that’s left to create is the custom control which will prompt the user for the photos they wish to upload.

1. Remain in **File View** for this part of the tutorial.
2. In the **Solution Explorer**, select the **Client** project’s ~/Scripts/ folder and select **Add** followed by **Existing Item**.
3. Browse to the location of the **photohelper.js** file and click **OK** to add the file to the project.
4. Open the **Client** project’s **default.htm** file in the code editor. Add a reference in the file to **photohelper.js**.

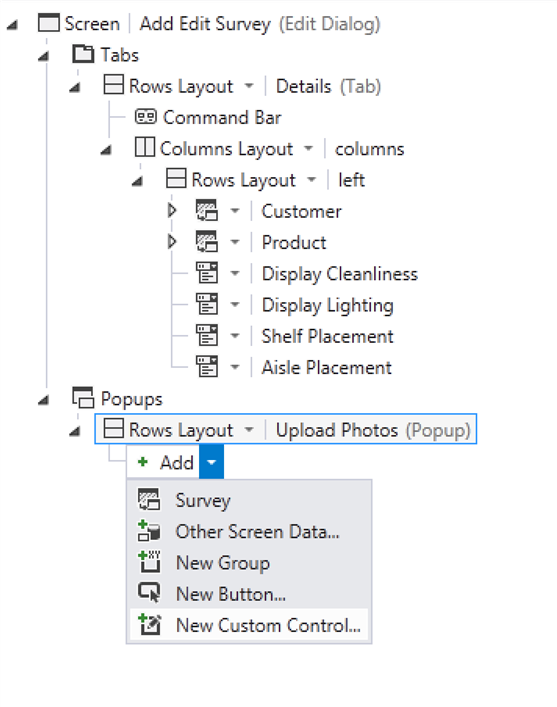
<script type="text/javascript" src="Scripts/msls-1.0.0.js"></script>

<script type="text/javascript" src="Scripts/sharepointauthhelper.js"></script>

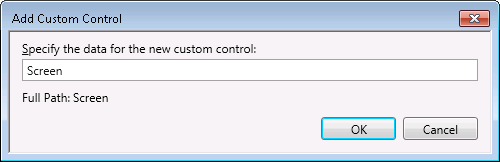
<script type="text/javascript" src="Scripts/photohelper.js"></script>

<script type="text/javascript" src="Scripts/Generated/data.js"></script>

1. In the **Solution Explorer**, switch back to **Logical View**.
2. Under the **Client** node, double-click the **AddEditSurveys** screen to open the screen designer.
3. In the screen designer, select the **Popups** node and click **Add Popup**. Rename the popup to **UploadPhotos** in the **Properties Window**.
4. Click **Add** and choose **New Custom Control…**



1. When the **Add Custom Control** dialog opens, set the data to **Screen** and click **OK** to close the dialog.



1. Select the custom control and in the **Properties Window**, rename the control to **UploadControl**. In addition, click the **Edit Render Code** hyperlink. When the **AddEditSurveys.js** file opens in the code editor, add the following code to the **UploadControl\_render** method:

var $uploadControlElement, uploadControlContentItem;

myapp.AddEditSurvey.UploadControl\_render = function (element, contentItem) {

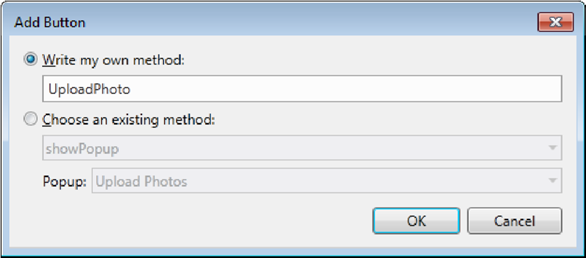
// Write code here.

$uploadControlElement = $(element);

uploadControlContentItem = contentItem;

};

1. Next, let’s add a button to the control to complete the upload. Select the **Upload** Photos rows layout node and click Add, then select **New Button…**
2. In the **Add Button** dialog select **Write my own method** and enter **UploadPhoto** in the text box and click **OK.**



1. Set the **Display Name** property of the button to **Upload** in the **Properties** window.
2. Double-click on the **UploadPhoto** node and add the following code to the button’s **execute** method.

myapp.AddEditSurvey.UploadPhoto\_execute = function (screen) {

if ($("#fileInput").val() != "") {

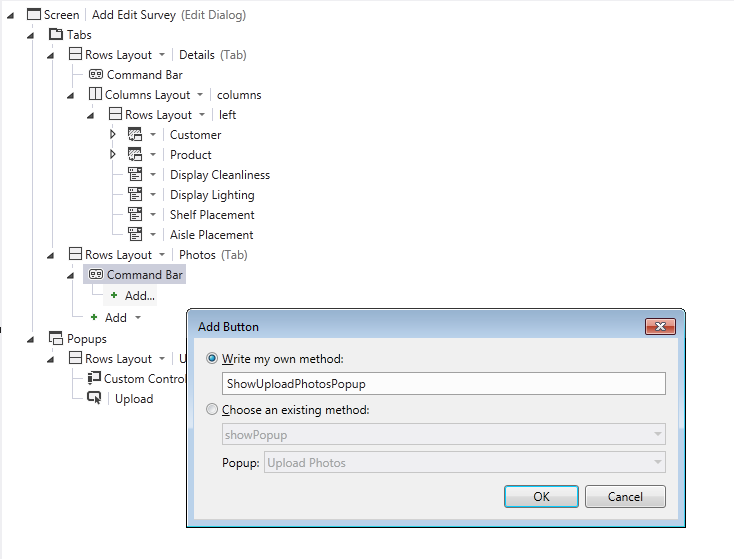
$.mobile.loading('show');

$("#uploadForm").submit();

}

};

1. Finally, we’ll add a button to the screen that opens the **UploadPhotos** popup.
   1. Select the **Tabs** node and click **Add Tab**. Rename the tab to **Photos** in the **Properties Window.**
   2. Under the **Command Bar** under the **Photos** tab, click the **Add…**
2. In the **Add Button** dialog:
   1. Choose the **Write my own method** option and enter **ShowUploadPhotosPopup** for the method name.
   2. Click **OK** to save changes and close the dialog.



1. In the **Properties Window**, set the button’s **Display Name** property to **Upload Photos**. Set the **Icon** type to **Add Picture.**
2. Open the **UploadPhotos\_execute** method in the code editor by double-clicking on the **Upload Photo** button in the screen designer. In the button’s **UploadPhoto\_execute** method, add the following code:

myapp.AddEditSurvey.ShowUploadPhotosPopup\_execute = function (screen) {

showUploadPopup(screen);

};

Notice that when a new photo is uploaded, this code, in the **photohelper.js** file, creates a new

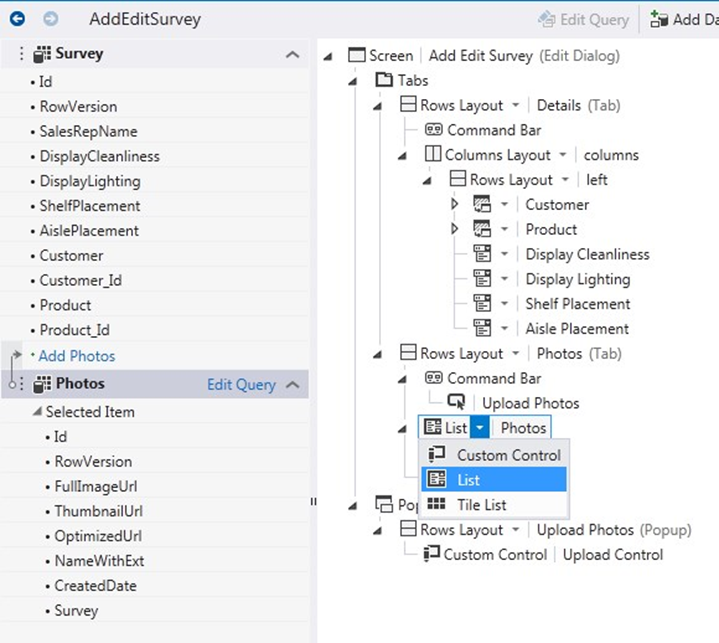
**Photo** entity in the intrinsic database that has property references to the full-sized, optimized and thumbnail image URLs in the Picture Library. This entity will be used in a later step to display the images in the Survey Application.

1. Build the application and verify there are no compile errors.

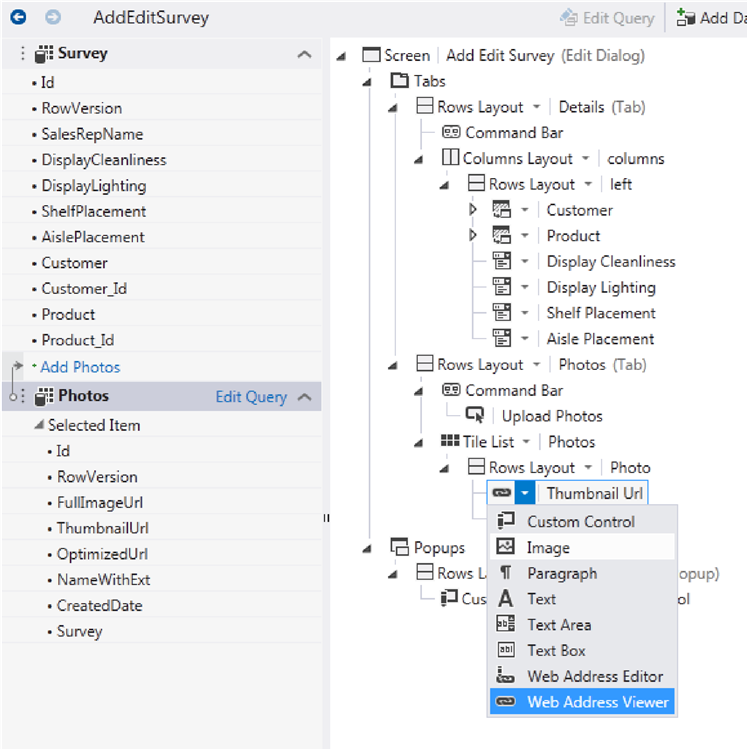
## 5.7 Add Screen Support for Displaying Photos

Next, let’s add support for displaying photos on the **AddEditSurveys** screen. For each uploaded photo, we’ll display the thumbnail sized image. When the user clicks the thumbnail image, we’ll open a popup that shows the larger, web optimized image.

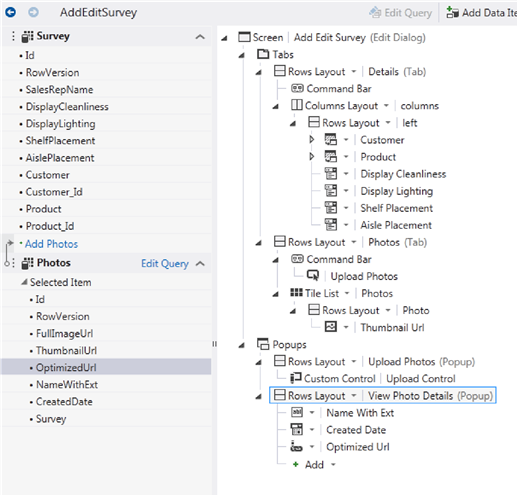
1. With the **AddEditSurveys** screen open in the screen designer, in the left-hand query pane click the **Add Photos** link to add the **Photos** query to the screen. Drag-and-drop the **Photos** query to the **Photos** tab and change the control type from **List** to **Tile List**. Under the **Photos Tile List** control, select the **Photos** **Rows Layout** control and set its **Width** and **Height** **Pixels** values to 60 in the **Properties Window**.



1. Delete the **Photo** properties so that only the **Thumbnail Url** is displayed. Change the **Thumbnail** **Url** control type from **Web Address Viewer** to **Image**. Note that the image control will automatically display the image based on the URL that the underlying property is set to.

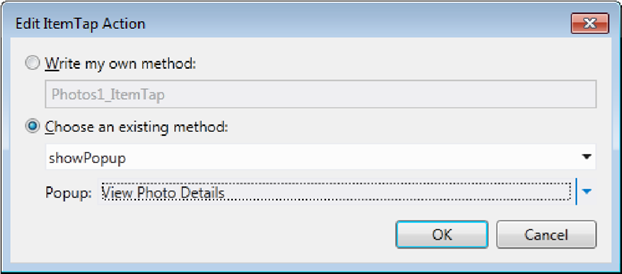


1. Next, let’s add a popup that will display the web optimized image. Select the **Popups** node and click **Add Popup**. Rename the popup to **ViewPhotoDetails** in the **Properties Window**. Drag and-drop the following **Photo** properties to the **ViewPhotoDetails** popup:
   1. **Name With Ext**
   2. **Created Date**
   3. **Optimized Url**



In addition:

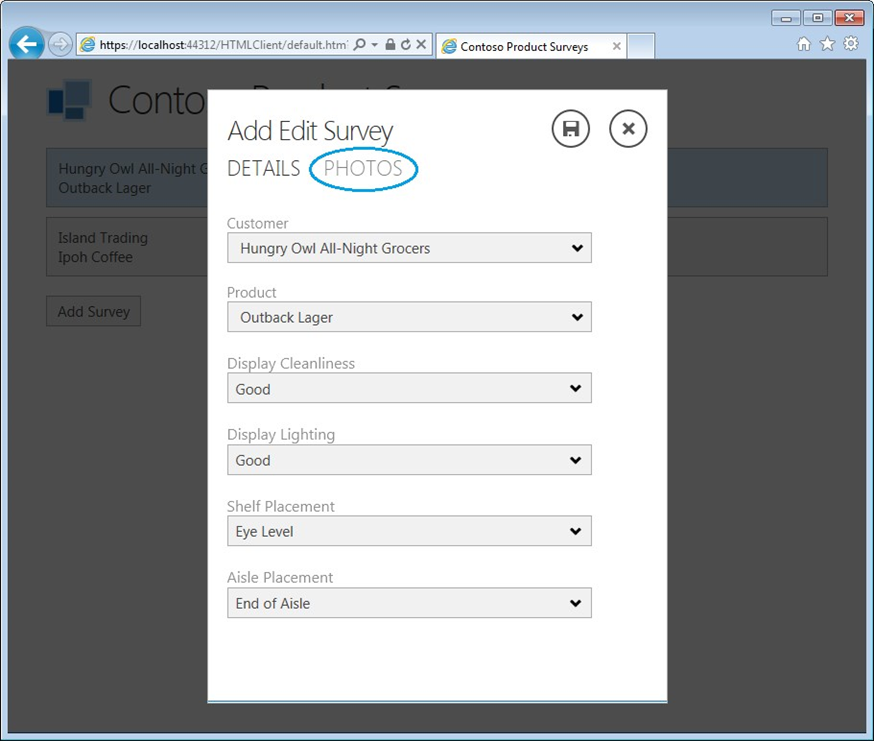
1. Change the **Name With Ext** control type from **Text Box** to **Text**.
2. Change the **Created Date** control type from **Date/Time Picker** to **Text**.
3. Change the **Optimized Url** control type from **Text Box** to **Image**.
4. In the **Properties Window**, set the **Optimized Url** control’s **Width** and **Height** to **Fit to Content**. Also set **Label Position** to **None**.
5. Select the **Photos Tile List** node and in the **Properties Window**, click the **Item Tap** hyperlink.
6. In the **Edit Item Tap** **Action** dialog:
   1. Choose the **existing method** option and select **showPopup.**
   2. Set the popup to **View Photo Details**.
   3. Click **OK** to save changes and close the dialog.



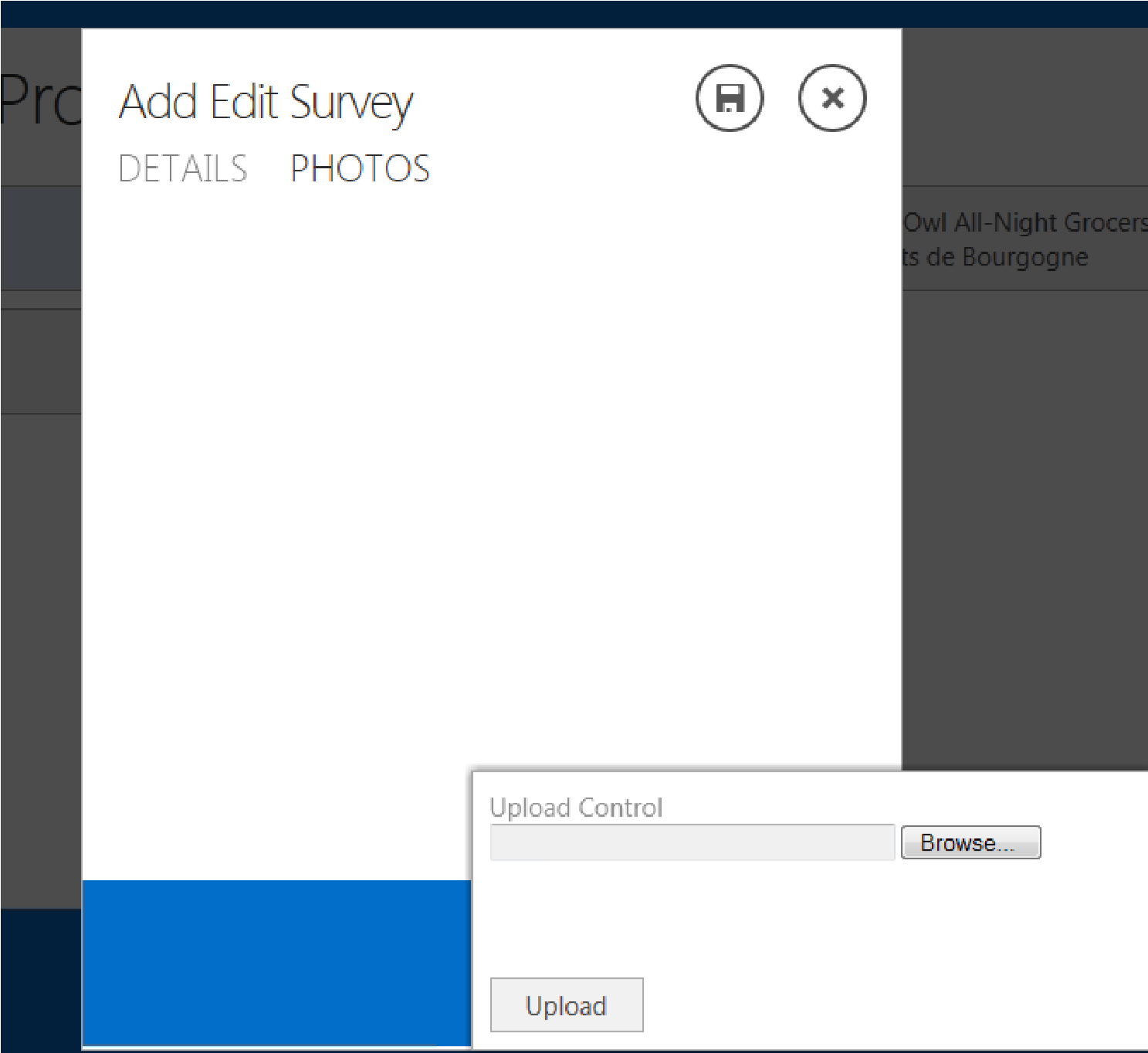
F5 Check Point: Build and Run the Application

To verify the additions we made to upload images, push the F5 key to build and run the application.

1. Click the survey assessment that you created earlier in the tutorial to open the **Add Edit Surveys** screen. Notice that there is now a **Photos** tab displayed.



1. Navigate to the **Photos** tab. At the bottom of the screen, click the **Upload Photos** button to open the custom image uploader. Browse to a photo on your local computer and click the **Upload** button. Once the image has finished uploading, the image thumbnail will be displayed in the Photos list.



1. Click the image thumbnail and verify the **View Photo Details** popup displays the web optimized image.
2. Stop the application from running by closing the web browser window.

## 5.8 Add Support for Deleting Photos

Next we’ll add logic in the save pipeline that removes the images from the **Photos** Picture Library when the corresponding **Photo** entity is removed.

1. In the **Solution Explorer** under the **Server** node, double-click the **Photos** table to open the entity designer.
2. Open the **Write Code** drop-down list and choose to override the **Photos\_Deleted** method. This method is executed within the server’s save pipeline each time that a **Photos** entity is deleted.
3. When the code editor opens for the **ApplicationDataService.[cs|vb]** file, insert the following code into the **Photos \_Deleted** method. This code calls the **PhotoListHelper ‘s DeletePhoto** method and passes in the app web’s **ClientContext** so that the CSOM API can be used to delete the corresponding image from the **Photos** Picture Library.

**VB:**

Private Sub Photos\_Deleted(entity As Photo)

Try

PhotoListHelper.DeletePhoto(entity.FullImageUrl, Application.SharePoint.GetAppWebClientContext())

Catch ex As Exception

End Try

End Sub

**C#:**

partial void Photos\_Deleted(SurveyPhoto entity)

{

try

{

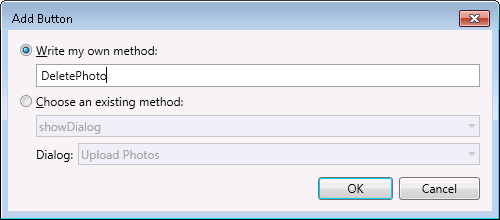
PhotoListHelper.DeletePhoto(entity.FullImageUrl, Application.SharePoint.GetAppWebClientContext());

}

catch (Exception) { }

}

1. Next, we’ll add a button used for deleting a photo. In the **Solution Explorer** under the **Client** node, double-click the **AddEditSurveys** screen to open the screen designer. Select the **View Photos** **Details** popup. Click **Add** and choose **New Button…**
2. In the **Add Button** dialog, choose the **Write my own method** option and set the name of the method to **DeletePhoto**.



1. Open the **DeletePhoto\_execute** method in the code editor by double-clicking on the **Delete Photo** button in the screen designer. In the button’s **DeletePhoto\_execute** method, add the following code:

myapp.AddEditSurveys.DeletePhoto\_execute = function (screen) {

deleteCurrentPhoto(screen);

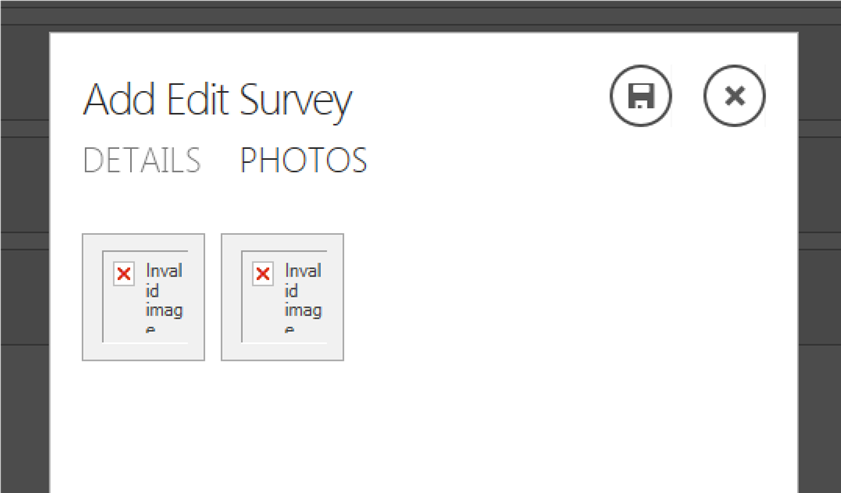
};

F5 Check Point: Build and Run the Application

To verify the delete behavior, push the F5 key to build and run the application.

1. Click the survey assessment that you created earlier in the tutorial to open the **Add Edit Surveys** screen.
2. Navigate to the **Photos** tab. Click the image thumbnail that you uploaded earlier to open the **View Photo Details** popup.
3. Click the **Delete** **Photo** button. Close the survey by tapping outside the popup.
4. Stop the application from running by closing the web browser window.

IMPORTANT: *Depending on the type of changes that you make to the Survey Application, F5 will automatically uninstall and reinstall the application. This includes removing and recreating the app web that contains the* ***Photos*** *Picture Library. As a result, if you’ve already uploaded survey photos and you make a change that causes the application to be uninstalled and reinstalled, the image URLs that are referenced in the application’s intrinsic database will become invalid. When this occurs, you’ll see an* ***Invalid Image URL*** *error in place of the photo. To fix this, simply delete any invalid photos and re-upload them.*

**

# Step 6: Publish the Finished App to SharePoint

So far you’ve only run the app in Debug mode, which uses SharePoint to provide authentication and redirects to the local computer’s instance of IIS Express. Your app is now complete and you can publish it to an on-premise SharePoint site, to Microsoft Azure, or to a third-party hosting site. After you publish your app, others can run it from SharePoint on their computers and mobile devices. See [How to: Publish a cloud business app to SharePoint](http://msdn.microsoft.com/en-us/library/office/dn632422(v=office.15).aspx).

After publishing, be sure to install the Customers and Products SharePoint lists on the new site by repeating the instructions in [4.1 Attach to Customer and Product SharePoint Lists](#_5.1_Attach_to).