# LightSwitch Mobile Business Apps Succinctly

by Jan Van der Haegen

Syncfusion | Technology Resource Portal

# LightSwitch Mobile Business Apps Succinctly

**By**

Jan Van der Haegen

Foreword by Daniel Jebaraj

Copyright © 2014 by Syncfusion Inc.
2501 Aerial Center Parkway
Suite 200
Morrisville, NC 27560
USA
All rights reserved.

**Important licensing information. Please read.**

This book is available for free download from www.syncfusion.com on completion of a registration form.

If you obtained this book from any other source, please register and download a free copy from www.syncfusion.com.

This book is licensed for reading only if obtained from www.syncfusion.com.

This book is licensed strictly for personal or educational use.

Redistribution in any form is prohibited.

The authors and copyright holders provide absolutely no warranty for any information provided.

The authors and copyright holders shall not be liable for any claim, damages, or any other liability arising from, out of, or in connection with the information in this book.

Please do not use this book if the listed terms are unacceptable.

Use shall constitute acceptance of the terms listed.

SYNCFUSION, SUCCINCTLY, DELIVER INNOVATION WITH EASE, ESSENTIAL, and .NET ESSENTIALS are the registered trademarks of Syncfusion, Inc.

**Technical Reviewer:** Alessandro Del Sole
**Copy Editor:** Darren West, content producer, Syncfusion, Inc.
**Acquisitions Coordinator:** Hillary Bowling, marketing coordinator, Syncfusion, Inc.
**Proofreader:** Tres Watkins, content development manager, Syncfusion, Inc.

# Table of Contents

# The Story behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President
Syncfusion, Inc.

## Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

## Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

## The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

## The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

## Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

## Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to "enable AJAX support with one click," or "turn the moon to cheese!"

## Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and "Like" us on Facebook to help us spread the word about the *Succinctly* series!

# About the Author

Jan Van der Haegen is a green geek who turns coffee into software.

He's a loving husband, LightSwitch lover, occasional blogger, former columnist for MSDN magazine online, senior technical architect for Trilogy Energy Software Inc, founder of a LightSwitch-oriented consultancy company, and he secretly dreams of becoming a professional goat herder one day.

Keep track of his latest adventures at http://switchtory.com/janvan.

# Foreword

This book is for me.

Not the current me, but the person I was a couple of months ago.

Up until then, I was frightened by the thought of having to write HTML/CSS/JavaScript applications instead of my comfort zone of C# and XAML, or the thought of writing applications for mobile devices.

Ignorance was to blame for this fear, mostly. At that time, I believed that making visually appealing applications in HTML5 meant you'd need to calculate correct points on a canvas using advanced geometry.  I believed that code written in JavaScript was more error-prone than code that had been compiled.  I believed that adapting to the screen size of both smartphones and tablets meant I would need to create different versions of every screen.  I believed CSS wasn't as maintainable as a good old Silverlight resource dictionary.

Honestly, I believed that the steep learning curve and the lack of professional tooling meant that it would take at least a couple of years before I would be writing my first HTML single-page application with a responsive design.

However, a couple of months ago I decided to give in to the lure of Visual Studio LightSwitch's newest addition: mobile business applications.  LightSwitch is a rapid application development environment within Visual Studio that abstracts technological choices behind easy to use editors, while still allowing the developer to take full control of any part of the application if needed.  This gave me the confidence to dive in head first and use LightSwitch to engage on a mobile-oriented HTML project without any prior knowledge of HTML5, CSS, or jQuery.

As it turned out, great tooling does already exist.  In fact, it is so good that using it actually makes that learning curve rather shallow.

This book is for me.

Not the current me, but the person I was a couple of months ago—eager to create mobile LOB apps to generate additional value for my customers, but lacking basic knowledge of HTML or JavaScript, and unaware of Microsoft's awesome tooling.

Maybe you are like the person I was.  In that case, this book is for you.

This book is your invitation, a challenge to leave your comfort zone and have your first mobile business application finished before the day is over.

I hope you enjoy.

Dedicated to my family for teaching me that by keeping our shoulders together

we can carry the weight of the world.

Especially dedicated to my loving wife Kundry, for keeping me focused on my goals,

and my goofy dog Mojo, for keeping me distracted.

# Introduction

The goal of this e-book is to teach you how to build a mobile-oriented single-page application (SPA) with Visual Studio LightSwitch by walking you through the process of building a CRM sample application.

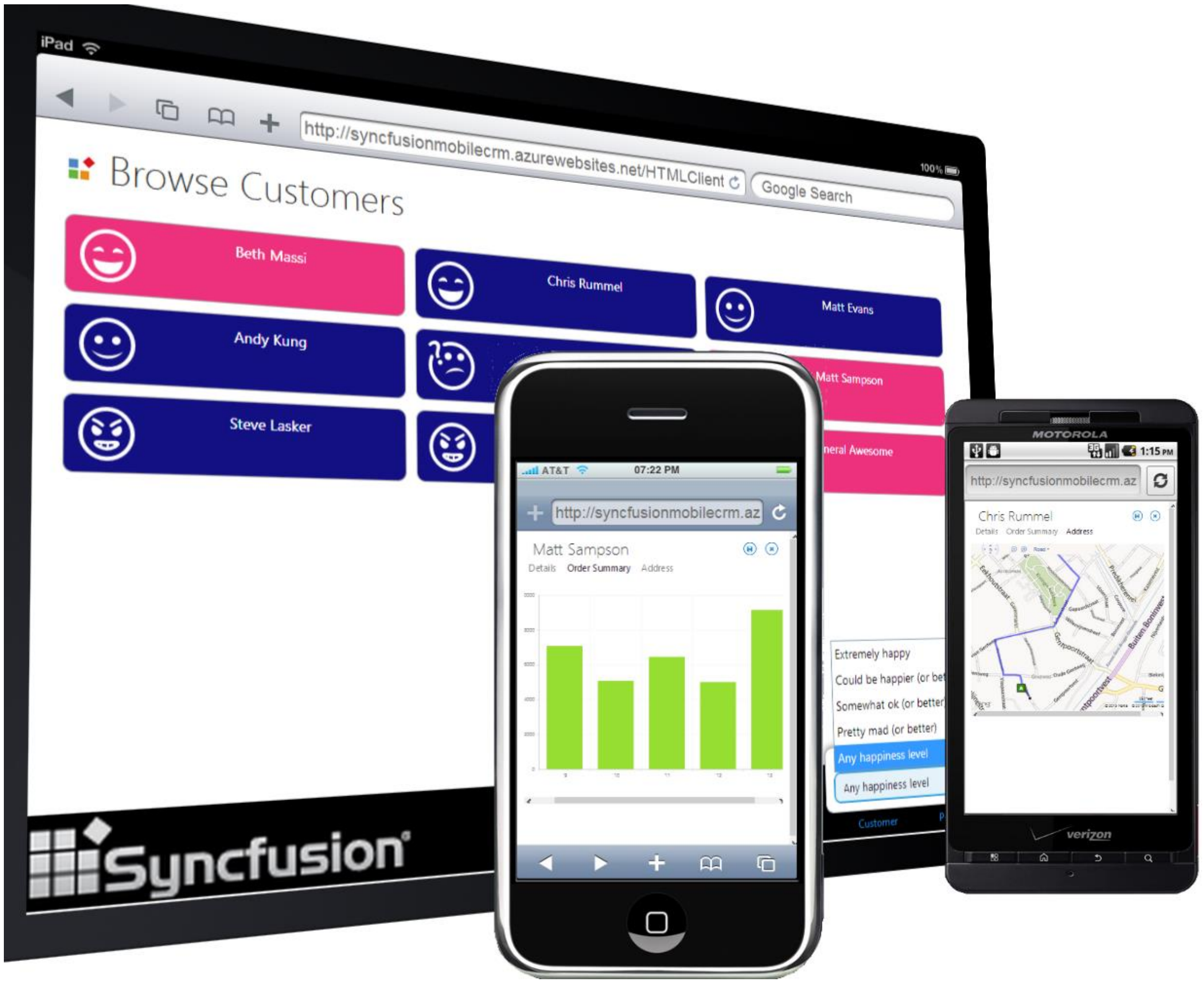


*Figure 1: A Mobile Business Application*

This application is an HTML5 single-page application that targets both smartphones and tablets, and creating it would take a LightSwitch developer a short time.

This book starts off with a very simple hello world application. After a bit of theory and LightSwitch vocabulary 101, you'll find a couple of hands-on chapters, each describing different aspects of a LightSwitch mobile business application in more depth: the database project, tweaking the visual aspects of the application, the programming model and techniques, and global styling. Finally, you'll deploy the application to the cloud, and I'll end the tour by providing some links to additional learning content.

I wrote this e-book with a particular audience in mind: one that is comfortable with a few general technological terms, but not fluent with the specific mobile and HTML technologies. If you are an expert, feel free to read through the book at a faster pace. I’m sure the LightSwitch Rapid Application Development approach will still be interesting and appealing.

If you are unfamiliar with the LightSwitch development experience, you might want to read *LightSwitch Succinctly* first, which explains at a slower pace the basic concepts of LightSwitch, and then you should continue reading this book, which dives in mobile apps specifically.

# Chapter 1  Creating a sample application

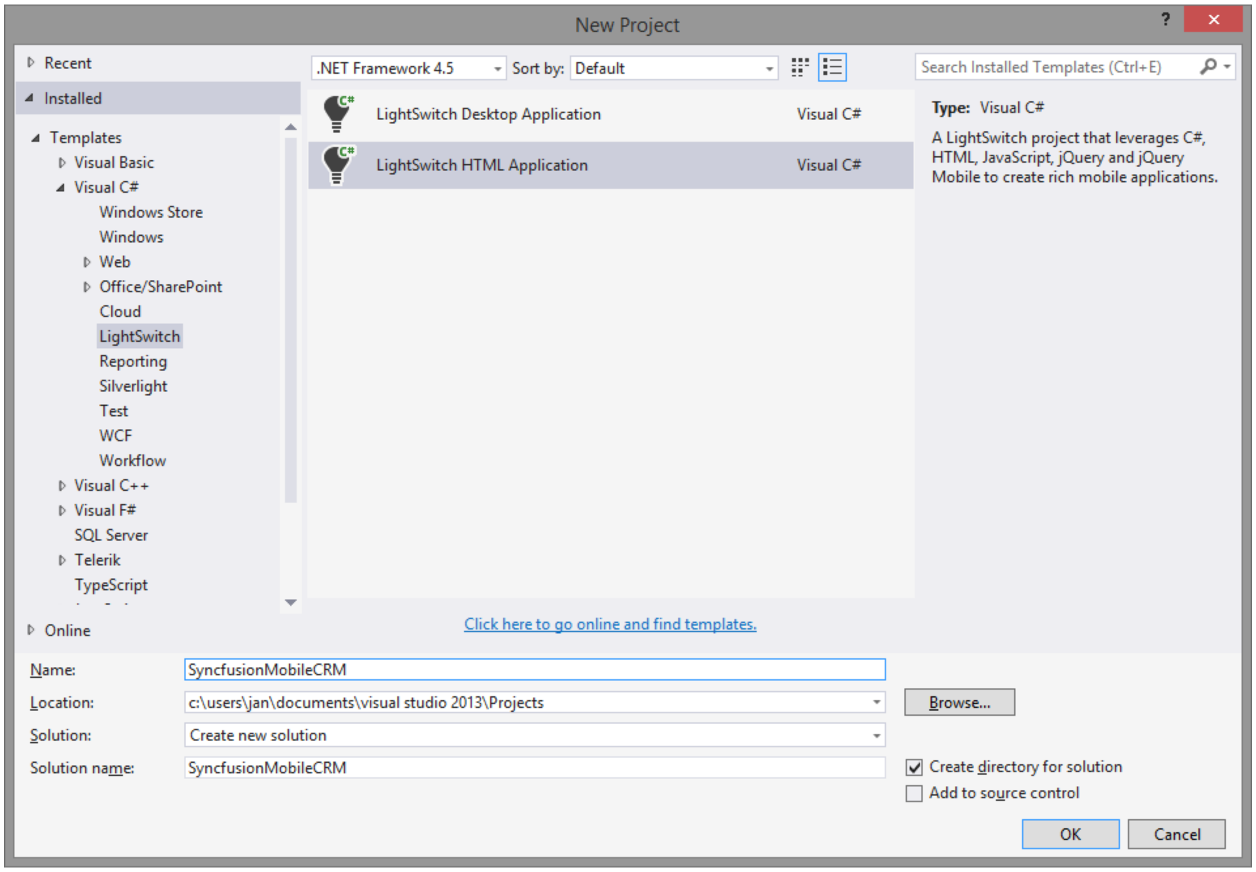
If you get stuck during this chapter or any part of the book, a completed sample application for Visual Studio 2013 can be downloaded from bitbucket.org/syncfusiontech/lightswitch-mobile-business-apps-succinctly.

LightSwitch mobile business applications are available in Visual Studio 2012 Professional or higher, with SP3 installed.

## Getting started

To create a hello world sample application, you start with file > new project.

The project type that you are looking for is a LightSwitch HTML Application, under the LightSwitch Template folder. I choose the C# version.



Type: Visual C#

A LightSwitch project that leverages C#, HTML, JavaScript, jQuery and jQuery Mobile to create rich mobile applications.

Name: SyncfusionMobileCRM

Location: c:\users\jan\documents\visual studio 2013\Projects

Solution: Create new solution

Solution name: SyncfusionMobileCRM

[✓] Create directory for solution

[ ] Add to source control

*Figure 2: New Project Wizard*

Initially, a LightSwitch solution will contain three projects: your LightSwitch project, the HTML client project, and a project for the server.

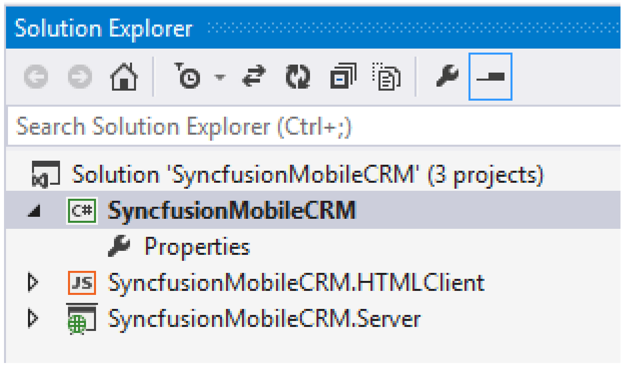

Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'SyncfusionMobileCRM' (3 projects)
SyncfusionMobileCRM
Properties
SyncfusionMobileCRM.HTMLClient
SyncfusionMobileCRM.Server

*Figure 3: A Shiny New LightSwitch Project in Solution Explorer*

This LightSwitch project's only responsibility is to keep the HTML client and the server project in sync where possible, so that you as a developer don't have to do this manually. Because this task is completely automated, the LightSwitch project itself appears empty.

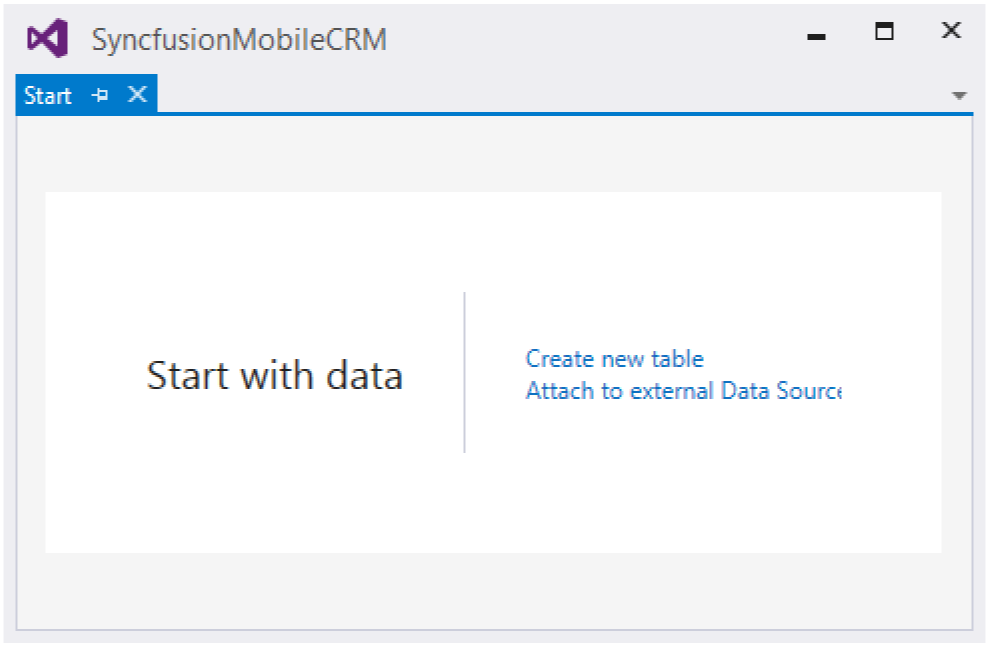


*Figure 4: LightSwitch Home Screen*

The LightSwitch home screen opened automatically. As suggested by the screen, applications begin with data. For this sample application, we will create a customer entity.

To add a customer entity, click **Create new table** in the home screen, or right-click on the server project in Solution Explorer and select **Add Table**.

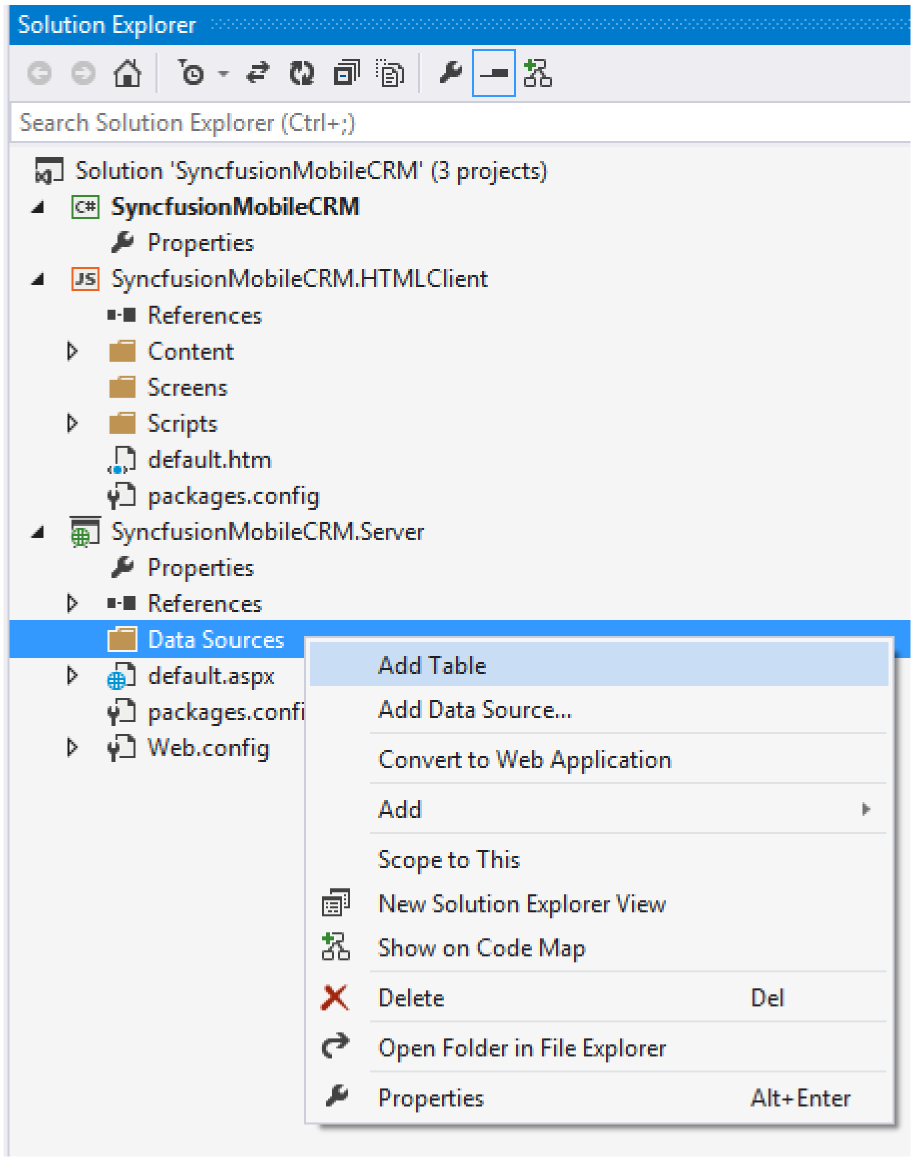


*Figure 5: Select Add Table from the context menu.*

This will open up the entity designer where you can design the customer model.

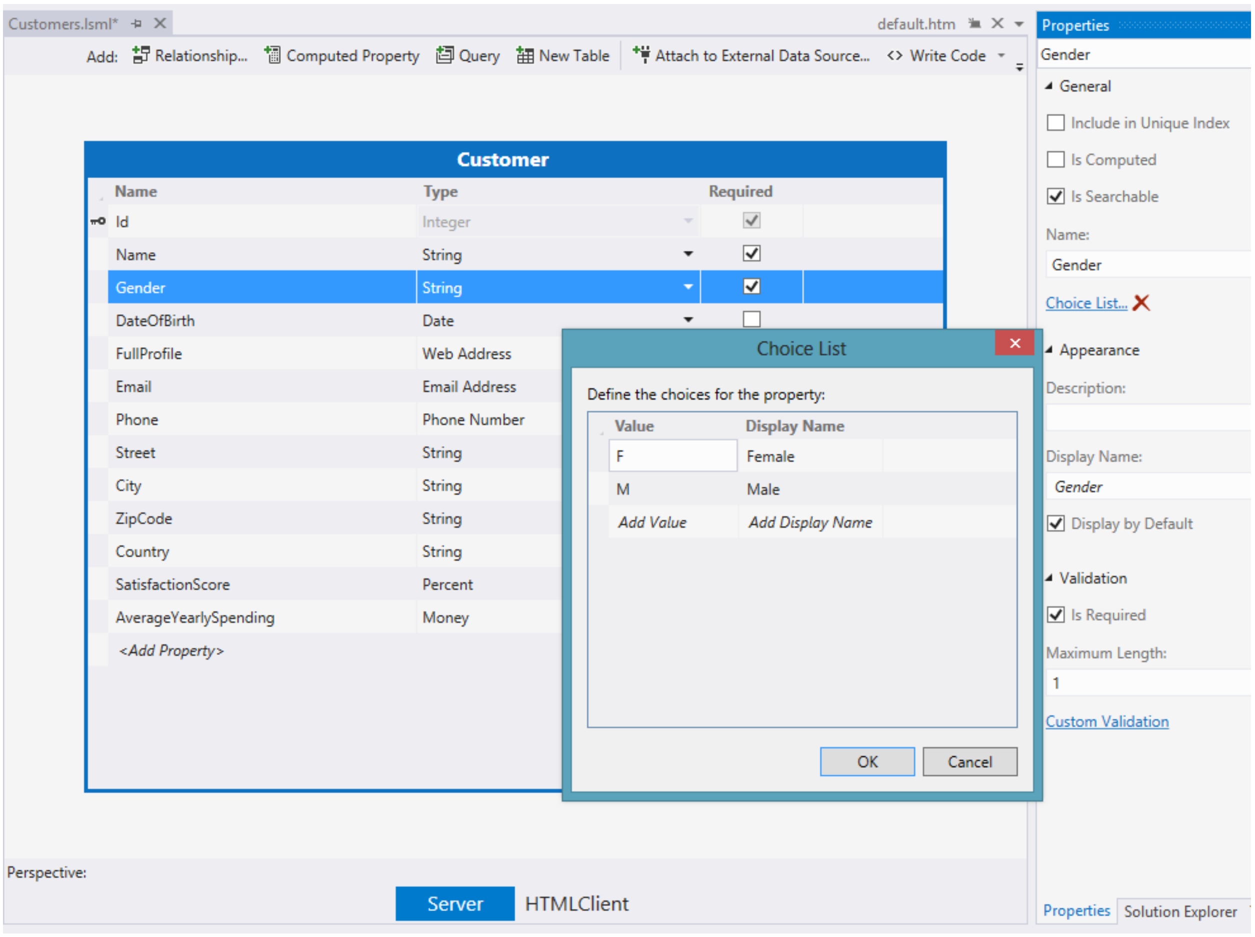
## Creating an entity

At a first glance, the entity designer looks like an expanded SQL database designer.

The entity designer isn't just about designing a backend data model though. A new SQL or SQL Azure database will be created with a customer table when you deploy, but the fields of your entity in the entity designer here are more than simple data types. They are true business types like phone number, e-mail address, web address, and person, each with built-in specialized validation rules, formatting, preferred controls, and extended options.

You can also use the properties in the properties window to change or add simple validation like setting minimum and maximum values, or even limiting the field to a strict choice list.

This won't suffice in an actual application, and you'll need custom initialization, validation, or authorization, which you can add by clicking **Write Code** from the perspective of the server or any of the clients.



| Name | Type | Required |
|---|---|---|
| Id | Integer | ✓ |
| Name | String | ✓ |
| Gender | String | ✓ |
| DateOfBirth | Date | |
| FullProfile | Web Address | |
| Email | Email Address | |
| Phone | Phone Number | |
| Street | String | |
| City | String | |
| ZipCode | String | |
| Country | String | |
| SatisfactionScore | Percent | |
| AverageYearlySpending | Money | |
| <Add Property> | | |

Define the choices for the property:

| Value | Display Name |
|---|---|
| F | Female |
| M | Male |
| Add Value | Add Display Name |

Properties: Gender

General
- [ ] Include in Unique Index
- [ ] Is Computed
- [✓] Is Searchable

Name: Gender

Choice List...

Appearance

Description:

Display Name: Gender

- [✓] Display by Default

Validation
- [✓] Is Required

Maximum Length: 1

Custom Validation

*Figure 6: Designing a Customer Entity*

A little effort in the entity designer is sufficient to design the customer entity. When the project is compiled, this will create the table structure, build a web service that exposes the middle tier, and prepare a JavaScript model in the HTML client.

## Creating screens

To finish the hello-world application, you'll need to create a few screens.

Right-click on the client project, and select **Add Screen** from the context menu.

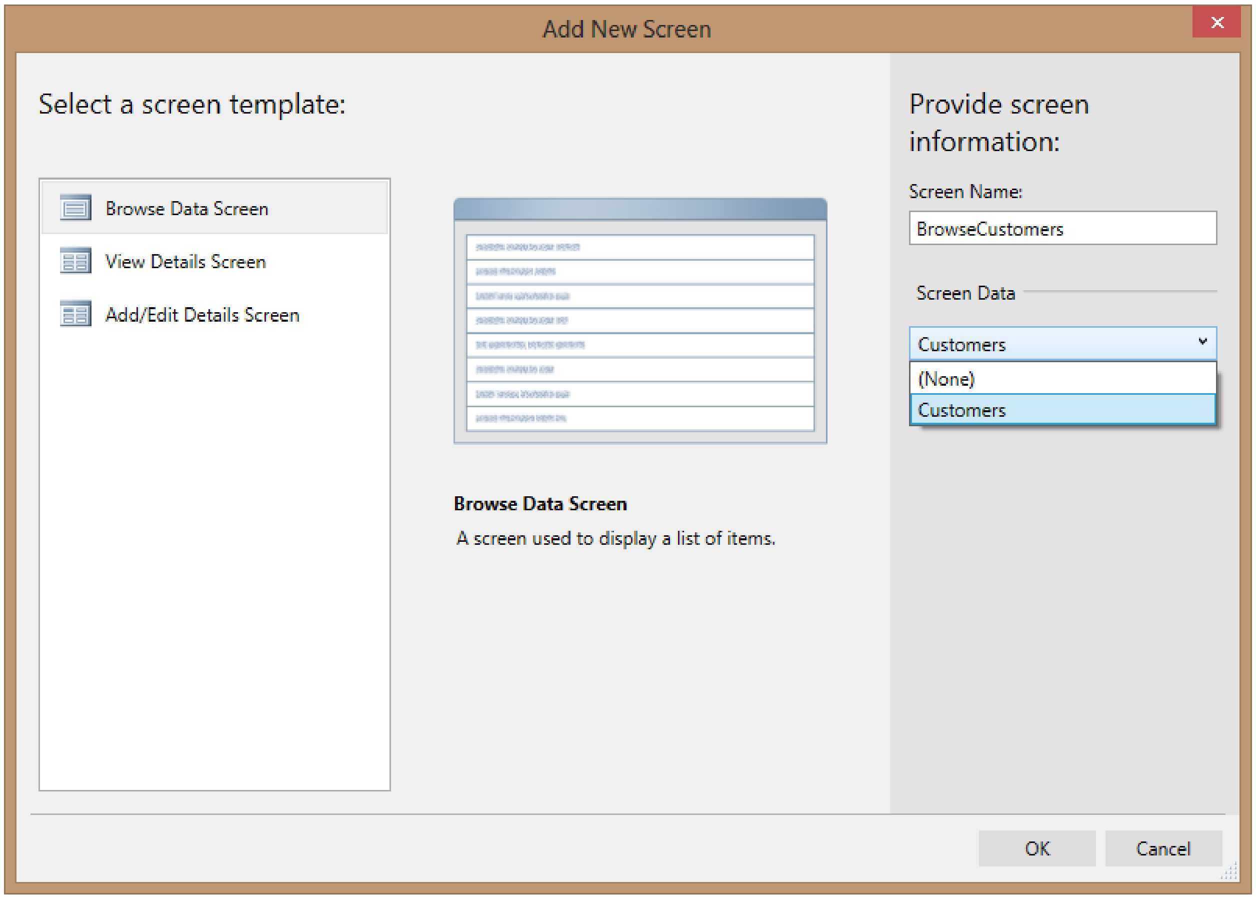


*Figure 7: New Screen Wizard*

The Add New Screen wizard will open with three built-in screen templates.

Select the **Browse Data Screen** template, and select **Customers** as the screen data from the dropdown box. You can leave the default screen name BrowseCustomers and click **OK**, which will generate the screen and open it in the screen designer.

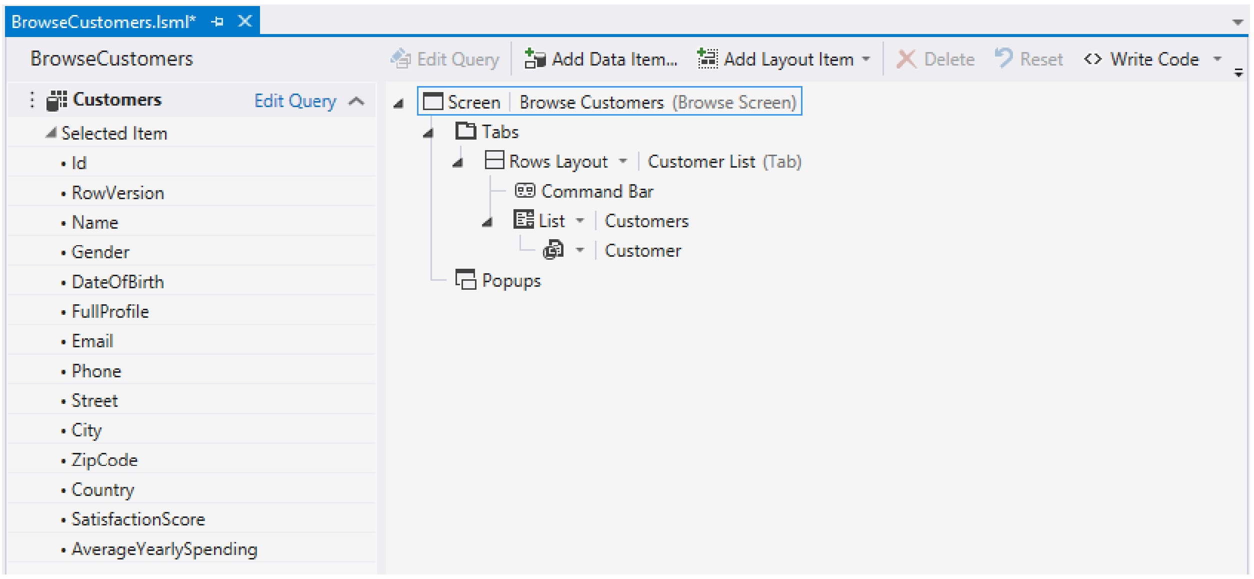
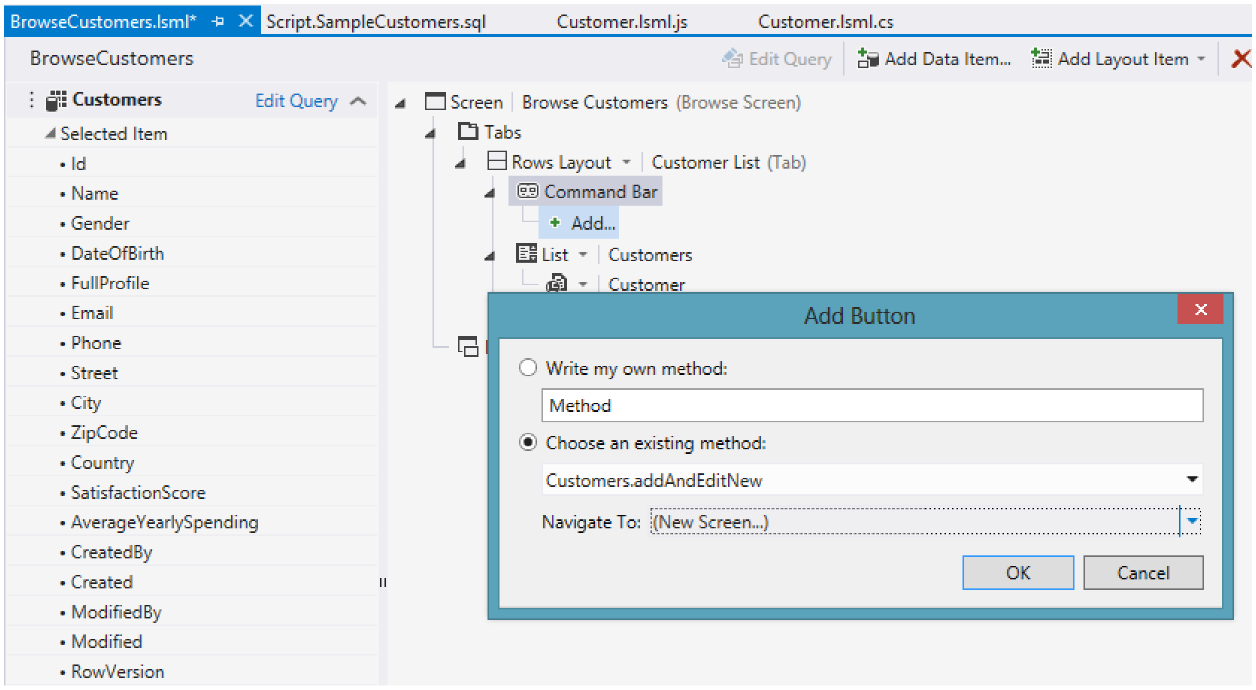


*Figure 8: The Screen Designer*

The left-hand side of the screen designer is called the screen view model (currently consisting only of a customer collection), and in the middle you'll find an abstraction of the view (a screen with a single tab that has an empty command bar and the customer List).

Since you start out with an empty database, you'll need to provide the end user some way to add new customers.

A common practice to do this is by adding an add button to the command bar.



Write my own method: Method

[✓ Choose an existing method:] Customers.addAndEditNew

Navigate To: (New Screen...)

*Figure 9: Add Button Wizard*

The add button wizard that opens gives you the option to write a custom method or to choose an existing method. The existing method combo box displays a couple of navigation options and a couple of actions you can perform on your customer collection. The number of existing methods in this combo box will grow as your application grows and your screens become more complex. For now you can bind your new Add button to a new customer and show a screen to edit that customer by selecting the Customers.addAndEditNew method.

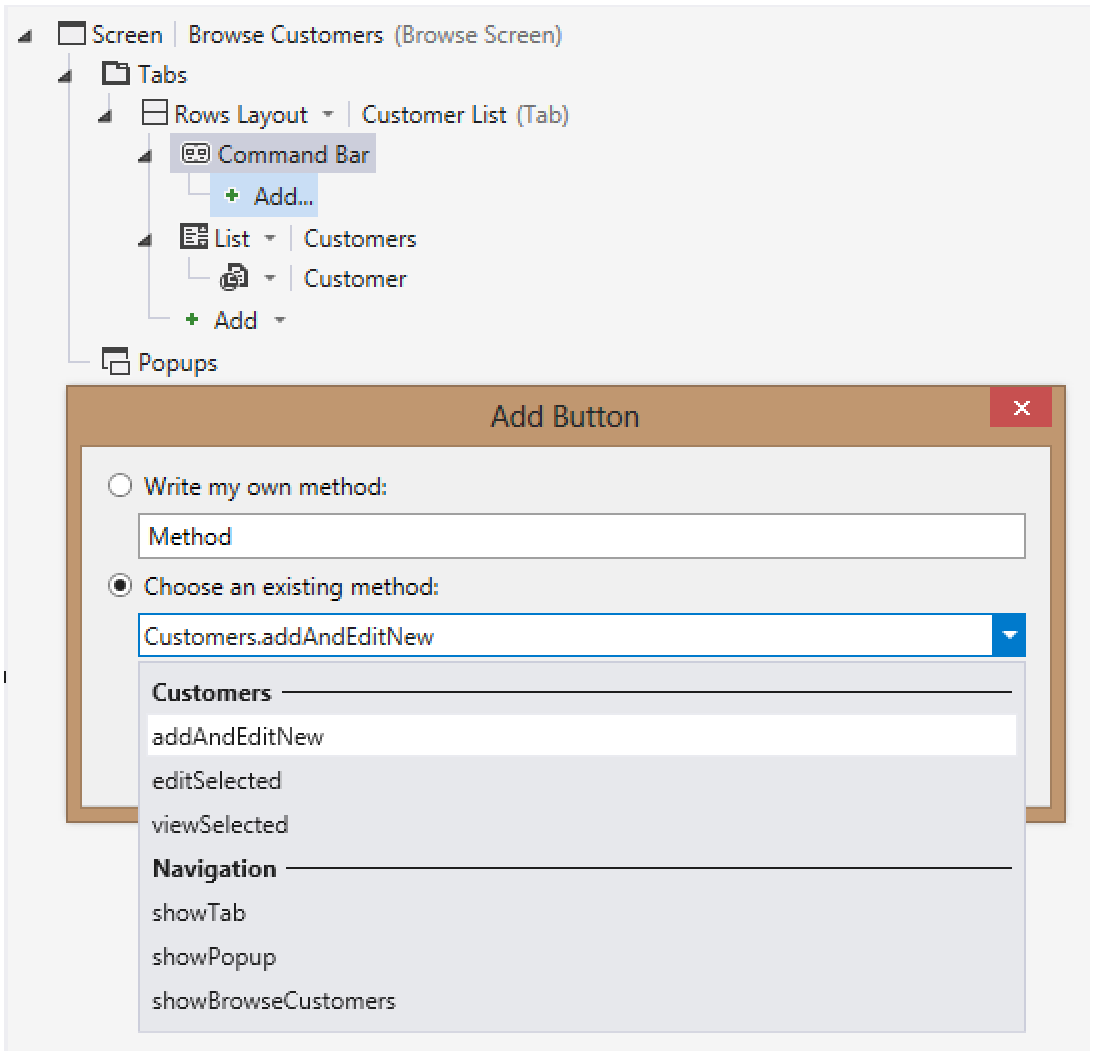


*Figure 10: The existing methods available are highly situational.*

When executed, the addAndEditNew method will create a new customer entity, add it to the visual collection on your browseCustomer screen, and then open up a detail screen to edit it. Because there is no screen to edit a customer yet, the wizard is suggesting navigating to a new detail screen.

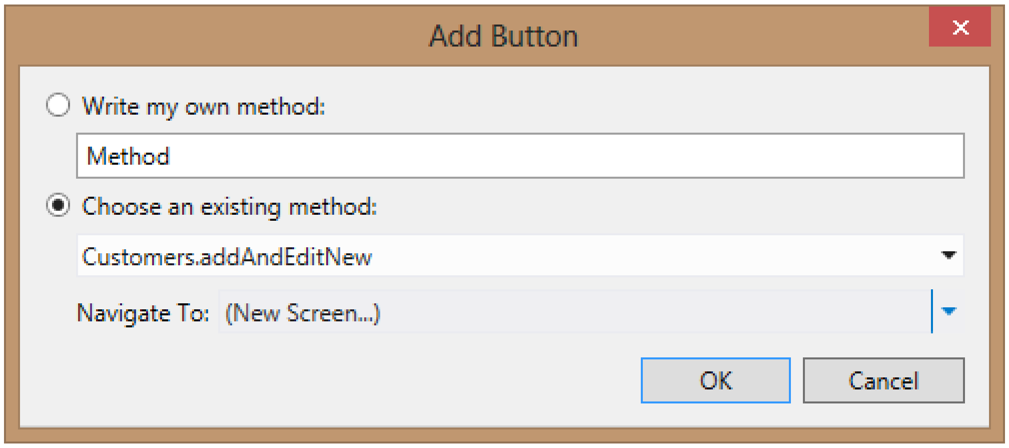


*Figure 11: Choosing an Existing Method*

Clicking **OK** will open up the Add New Screen wizard to help generate your detail screen.

After accepting that wizard, you will have two screens in your application: an all customers overview screen (browseCustomers) and a customer details screen (customers). The first overview screen in the application will be the Home screen. It opens automatically when the applications starts.

Now that you have a detail screen to create a customer, why not reuse that screen to edit a customer as well? To do this, you have to bind the tab gesture on the list (this occurs when the end user taps or clicks on an item in the list) to open the detail screen.

Open the browseCustomers screen in the screen designer, select the List of customers node, and from the properties window, select the Item Tap action, which is currently set to None.

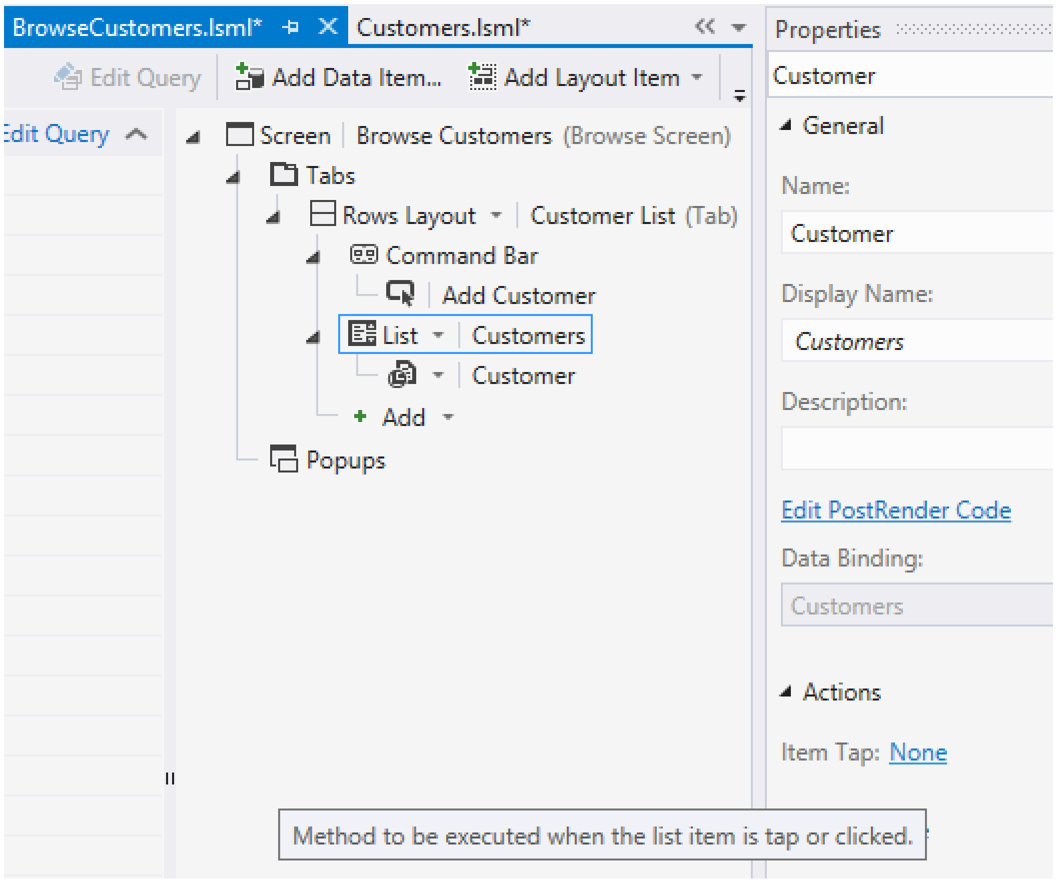
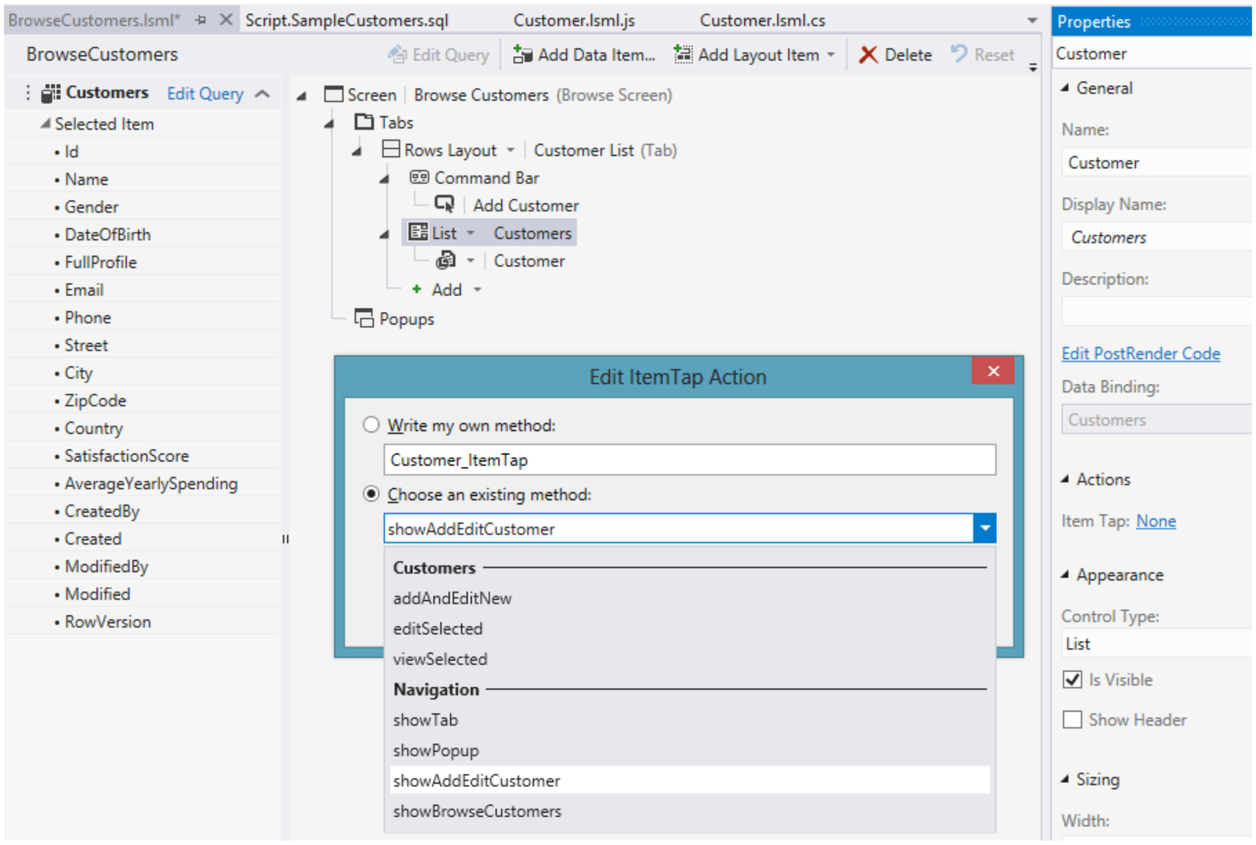


*Figure 12: Responding to Touch Gestures*

In the Add ItemTap Action dialog that appears, there will now be an additional navigation method: opening the AddEditCustomer screen.



Edit ItemTap Action

( ) Write my own method:

Customer_ItemTap

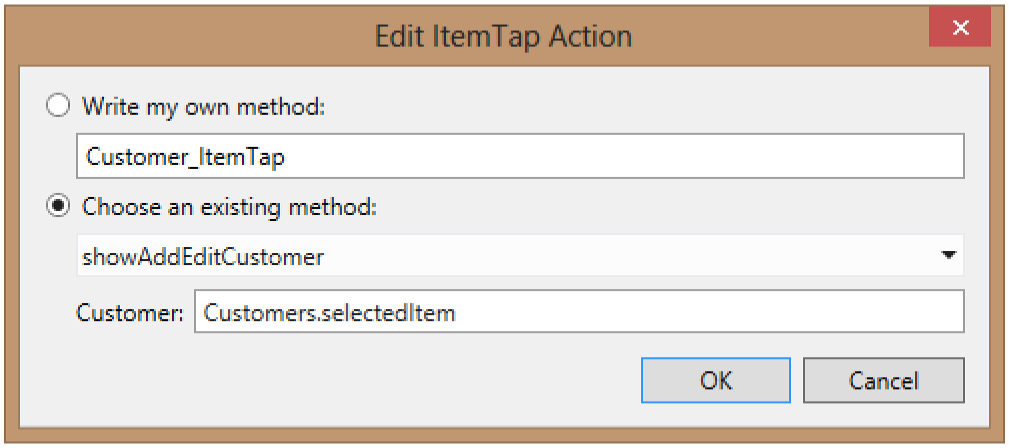
(●) Choose an existing method:

showAddEditCustomer

**Customers**
addAndEditNew
editSelected
viewSelected
**Navigation**
showTab
showPopup
showAddEditCustomer
showBrowseCustomers

*Figure 13: Navigating to the Newly Created Screen*

This detail screen requires you to define parameters to determine which customer's details to display. To pass the currently selected item as customer details, type Customers.selectedItem. IntelliSense is provided everywhere in the LightSwitch experience, including wizards.



Edit ItemTap Action

( ) Write my own method:

Customer_ItemTap

(●) Choose an existing method:

showAddEditCustomer

Customer: Customers.selectedItem

OK | Cancel

*Figure 14: Passing the Currently Selected Item as a Customer Parameter*

## Exploring the application

Press Ctrl+F5 to build and run the application.

The Browse Customers screen you created first will automatically open. Since there are no customers in the database, it'll be rather empty.

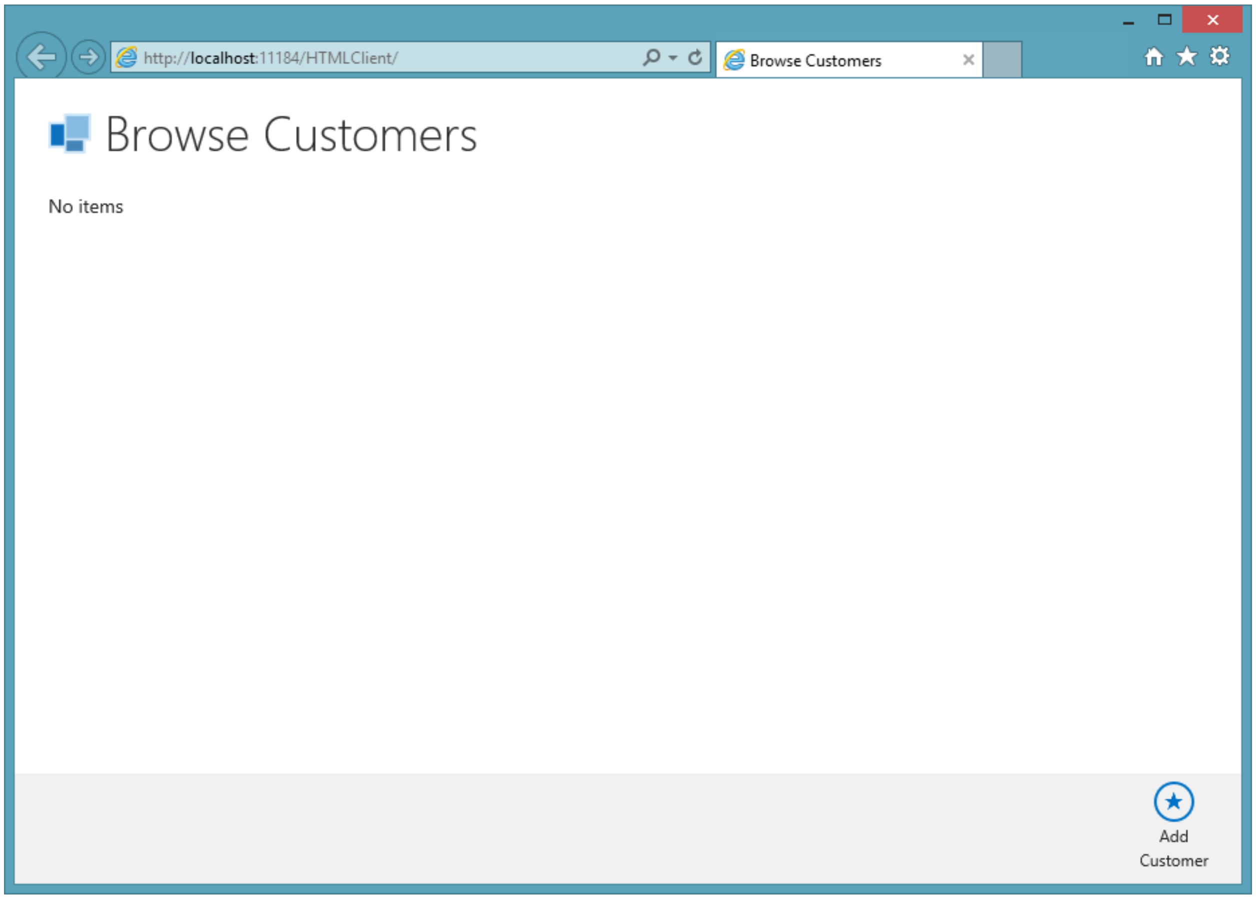


*Figure 15: An Empty but Fully Functional Application*

Hitting the Add Customer button from the Command Bar (the gray footer) will add a new customer to the list and open the Add Edit Customer screen as a modal dialog.

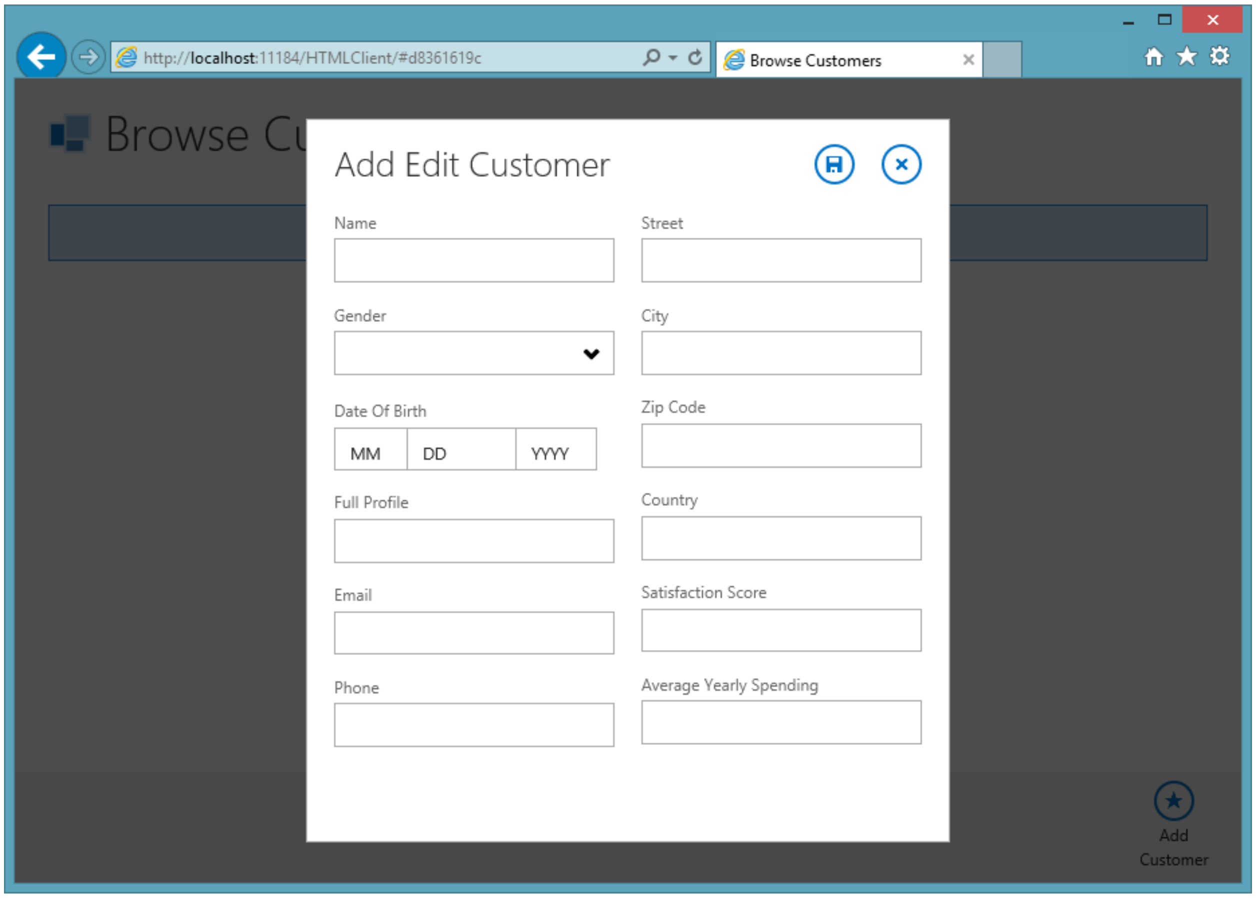
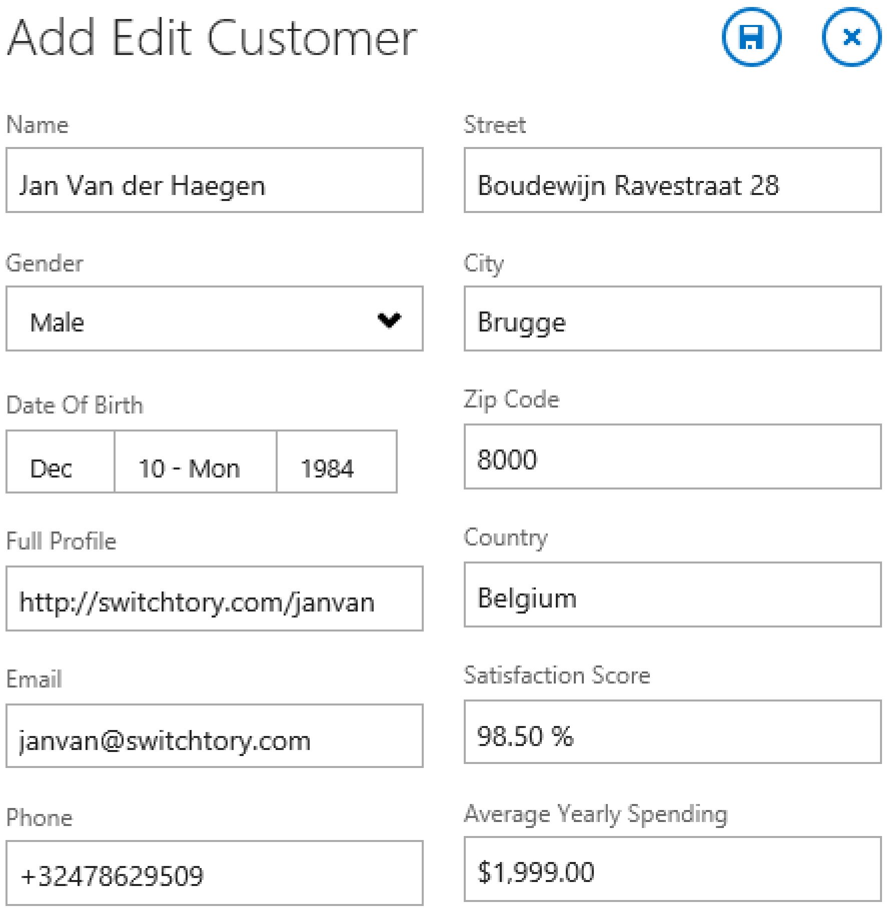


*Figure 16. Add Edit Customer*

As you proceed to fill in the details of your first customer, you'll notice how the application respects the way that you have modeled your customer entity in the entity designer. The appropriate business type for each property will be used to infer the best way to visualize a particular field.



Add Edit Customer

| | |
|---|---|
| Name | Jan Van der Haegen |
| Gender | Male |
| Date Of Birth | Dec 10 - Mon 1984 |
| Full Profile | http://switchtory.com/janvan |
| Email | janvan@switchtory.com |
| Phone | +32478629509 |
| Street | Boudewijn Ravestraat 28 |
| City | Brugge |
| Zip Code | 8000 |
| Country | Belgium |
| Satisfaction Score | 98.50 % |
| Average Yearly Spending | $1,999.00 |

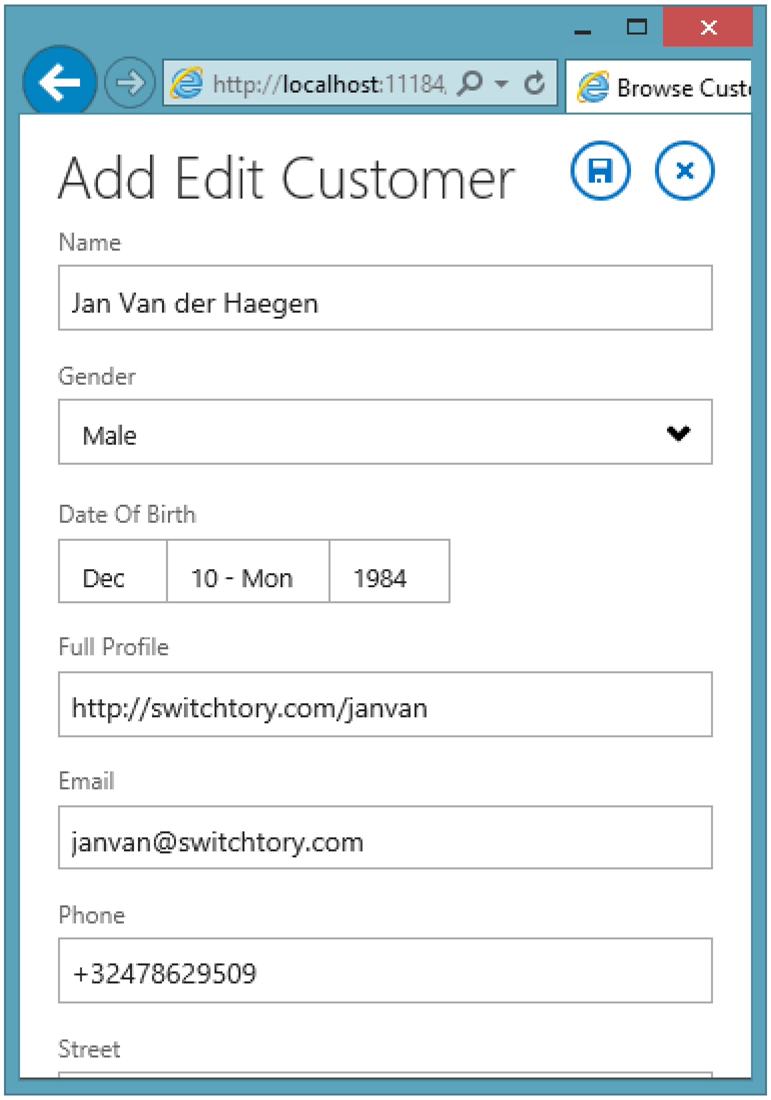
*Figure 17: Create a New Customer*

## Creating a new Customer

### Some tech talk

Congratulations, you just created a fully functional application with only half a cup's worth of coffee. I'm sure the app looks good, and I'm happy to report it's quite the technical masterpiece too.

One of the technical details you might miss at first glance is the responsive design. Make the browser smaller and larger, and you'll notice how the application carefully chooses which elements to resize and which to keep at a fixed size to preserve as much screen estate as possible for the content that matters most. If you make the browser really small, you'll notice how the application even reduces the number of columns from two to one to avoid a messy layout on smaller form factors like smartphones.



Add Edit Customer

Name

Jan Van der Haegen

Gender

Male

Date Of Birth

Dec | 10 - Mon | 1984

Full Profile

http://switchtory.com/janvan

Email

janvan@switchtory.com

Phone

+32478629509

Street

*Figure 18: The layout is updated to adapt to the device's screen size.*

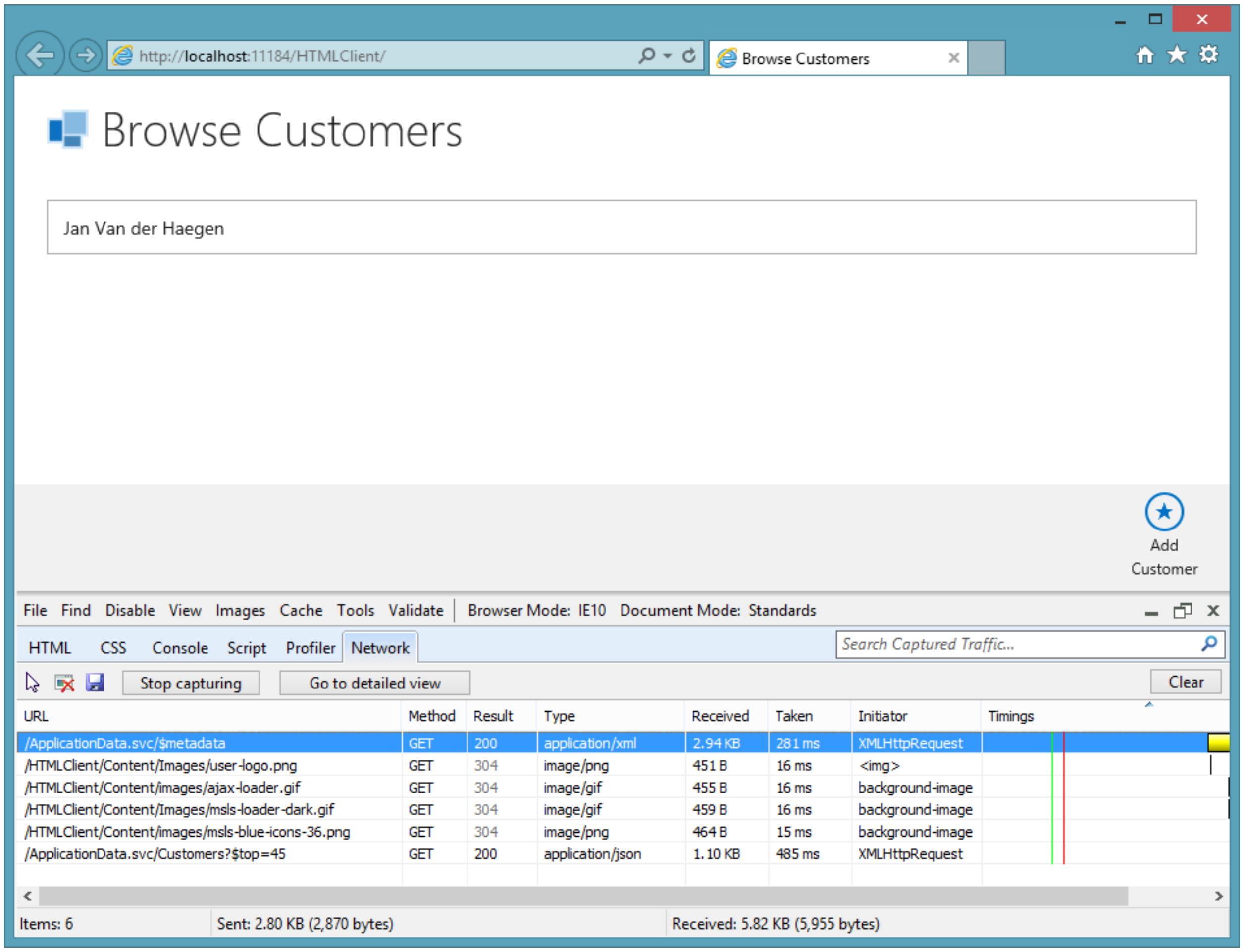
Program Manager Heinrich Wendel has a post explaining the team's envisioned strategy for designing an application with an adaptive design for multiple form factors which you can find at http://blogs.msdn.com/b/lightswitch/archive/2013/03/12/designing-for-multiple-form-factors.aspx.

A second technical detail about the application you might miss at first glance, especially with only one record in the database, is its built-in paging mechanism.

LightSwitch exposes the service layer to the HTML client by means of a data service that implements the OData protocol. OData is an open standard that centers on using HTTP verbs (GET, POST, etc) and a URI to identify resources, in combination with a fixed number of operators to request filtering, sorting, transformations, etc.

The HTML client will use the OData Top operator to instruct the server to return only the first 45 records. You can verify this using your preferred network traffic-capturing tool (like F12 in Internet Explorer) and by capturing the communication between the Browse Customers screen in the HTML client and the OData service.

The OData protocol, examples, and ecosystem can be found at http://OData.org.


http://localhost:11184/HTMLClient/
Browse Customers
Browse Customers
Jan Van der Haegen
Add Customer
File Find Disable View Images Cache Tools Validate Browser Mode: IE10 Document Mode: Standards
HTML CSS Console Script Profiler Network
Search Captured Traffic...
Stop capturing
Go to detailed view
Clear

| URL | Method | Result | Type | Received | Taken | Initiator | Timings |
|---|---|---|---|---|---|---|---|
| /ApplicationData.svc/$metadata | GET | 200 | application/xml | 2.94 KB | 281 ms | XMLHttpRequest | |
| /HTMLClient/Content/Images/user-logo.png | GET | 304 | image/png | 451 B | 16 ms | &lt;img&gt; | |
| /HTMLClient/Content/images/ajax-loader.gif | GET | 304 | image/gif | 455 B | 16 ms | background-image | |
| /HTMLClient/Content/Images/msls-loader-dark.gif | GET | 304 | image/gif | 459 B | 16 ms | background-image | |
| /HTMLClient/Content/images/msls-blue-icons-36.png | GET | 304 | image/png | 464 B | 15 ms | background-image | |
| /ApplicationData.svc/Customers?$top=45 | GET | 200 | application/json | 1.10 KB | 485 ms | XMLHttpRequest | |

Items: 6 Sent: 2.80 KB (2,870 bytes) Received: 5.82 KB (5,955 bytes)

*Figure 19: The application pages the data.*

If there are more than 45 records, the next batch of 45 will be fetched asynchronously when the user scrolls or swipes down.

Filtering is done by SQL Server for best performance and can be turned off or customized (different page size) from the properties window in the screen designer.

Feel free to play around with the application, but don't get too attached to the customers you create, as we'll soon replace them with sample data using a SQL script. In the next section, you will be introduced to the advanced concepts of LightSwitch development.

# Chapter 2  Exploring LightSwitch

## What is LightSwitch?

LightSwitch does not have an official definition, and because of how much it has grown since its first release and the broad array of software development challenges that it can help with, it is not an easy task to define.

For the purpose of this e-book, LightSwitch is a combination of frameworks and integrated tooling that aids professional developers building highly specialized LOB applications (stand alone or as part of a bigger ecosystem) in rapid release cycles. It does not require compromise on interoperability or flexibility, and instead enforces industry proven standards and patterns and promotes scalable architectural designs.

## What are LOB applications?

Line-of-business (LOB) applications are applications that are written to be privately used by a particular company. Estimates say that about 90% of all software written today is LOB applications. Each LOB app is unique to the business process, the company culture, and the geographical location where it is used. However, there are some common denominators.

Most LOB applications model the real world in domain-specific software structures called, appropriately, models. LightSwitch developers refer to these models as **Entities**. Entities need to be stored so that they can be reused between runs, shared between different end users, or even exchanged between different applications. Some LOB applications have a set of data so unique that it can be stored in a data store, called the **Intrinsic data source**, that is unique to the application. Often though, LOB applications are part of a larger ecosystem and will reuse already existing data sources.

**Data source** doesn’t necessarily mean database. Entities can be stored in and retrieved from a variety of sources: a new or existing database (from another active or legacy LOB application), web services (such as SharePoint lists, TFS work items, other LOB applications, and third-party sources) or even files on a network share.

Entities represent real-life objects (or parts thereof). They are not mere data structures, but intelligent binary creations that encapsulate business logic. Some business logic will be unique to the entity or specific property, while other logic might be quite common in the particular business or even in general. LightSwitch will refer to this common logic as **Business types**. Business types are a superset of data storage types (string, int, binary, etc.) that have a unique set of business logic and validation rules, such as Money, E-mail address, and Phone number.

Besides business logic limitations, all entities will suffer from system limitations imposed by the backing data source like the maximum length of a particular field, the number of decimal fractions that can be stored, etc.

Entities aren't isolated cells, but instead interact with each other according to a set of balanced and protected **Relationships**.

It's best practice to enforce those relationships where possible, but this isn't always possible in the data source. Besides, two entities might have a relationship according to the business regardless of being persisted in different, isolated data stores.

Almost all LOB applications are designed for multi-user purpose. This means that the software cannot be a monolith of bytes on a single pc, but instead should be vertically sliced per reusability and horizontally sliced per deployment location. These horizontally sliced blocks are often referred to as tiers. Multi-user LOB applications commonly consist of a Data tier, Logic tier, and Presentation tier. In the vocabulary of a LightSwitch developer, these three types of tiers in such a multi-tier architecture are known as **Data sources**, **Middle Tier,** and **Clients**. Entities will really exist in all different tiers. Ideally, their business logic and their relationships are enforced on each tier as well, increasing the way a layer is **autonomous**, and thus increasing the overall **reusability** of the layer.

Each tier is structured internally according to specific architectural or library **design patterns**. A client tier, for example, can be structured according to the **Model-View-ViewModel (MVVM)** pattern where the View Model layer prepares the Models and reusable commands for easier consumption by the presentation layer (View). End users of an LOB application are often non-tech savvy people who think about their work in specific business terms. Good LOB application developers will understand this and not limit the end user to only the CRUD commands (Create, Read, Update, and Delete) or to only displaying all data in grids. *(I have gridophobia, by the way.)* Good LOB application developers will understand the specific use of the application and use typography, colors, custom layouts, and specialized controls tailored to the particular business process and particular end.

This also includes an **adaptive design** (the layout or controls are changed depending on the type or screen factor of the device), utilizing device-specific capabilities or supporting speech or touch gestures (if applicable) and in some cases support for multiple languages.

Finally, when new features are completed by the developer, the code is pushed to source control, compiled if needed, and then deployed. The release cycle is best kept as short as possible so feedback can be incorporated early and often. A good **deployment model** should be **highly automated** in order not to slow down the release cycle.

End users then install the application (or simply browse to the web application) and start using what is accessible to them. There's a magnitude of known **authentication** and sometimes complex **authorization** models to make sure data and business processes are granted only to users with the appropriate authorizations.

## The LightSwitch IDE

The LightSwitch IDE mainly utilizes specialized designers to design Entities, Queries, and Screens. These designers abstract the technological aspects and remove a lot of the tedium involved to allow you as a professional developer to focus only on the business aspects initially.

The entity designer is automatically opened if you start the application like you did in the previous chapter: by right-clicking on the data sources folder of the server project and selecting Add Table.

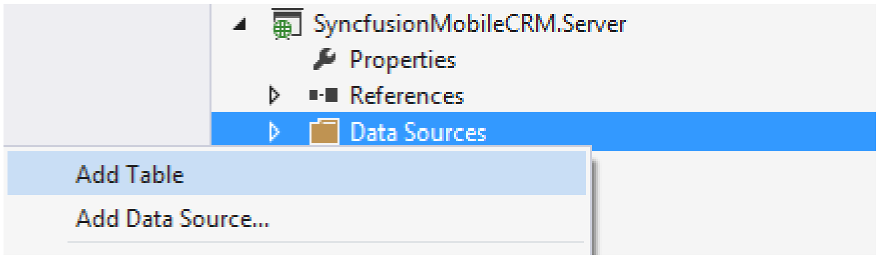


*Figure 20: Data Sources Context Menu*

This command automatically caused LightSwitch to add a new data source called ApplicationData.

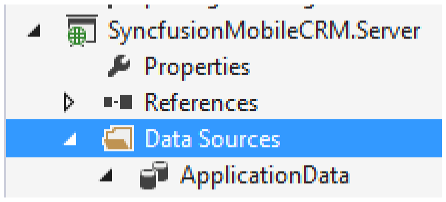
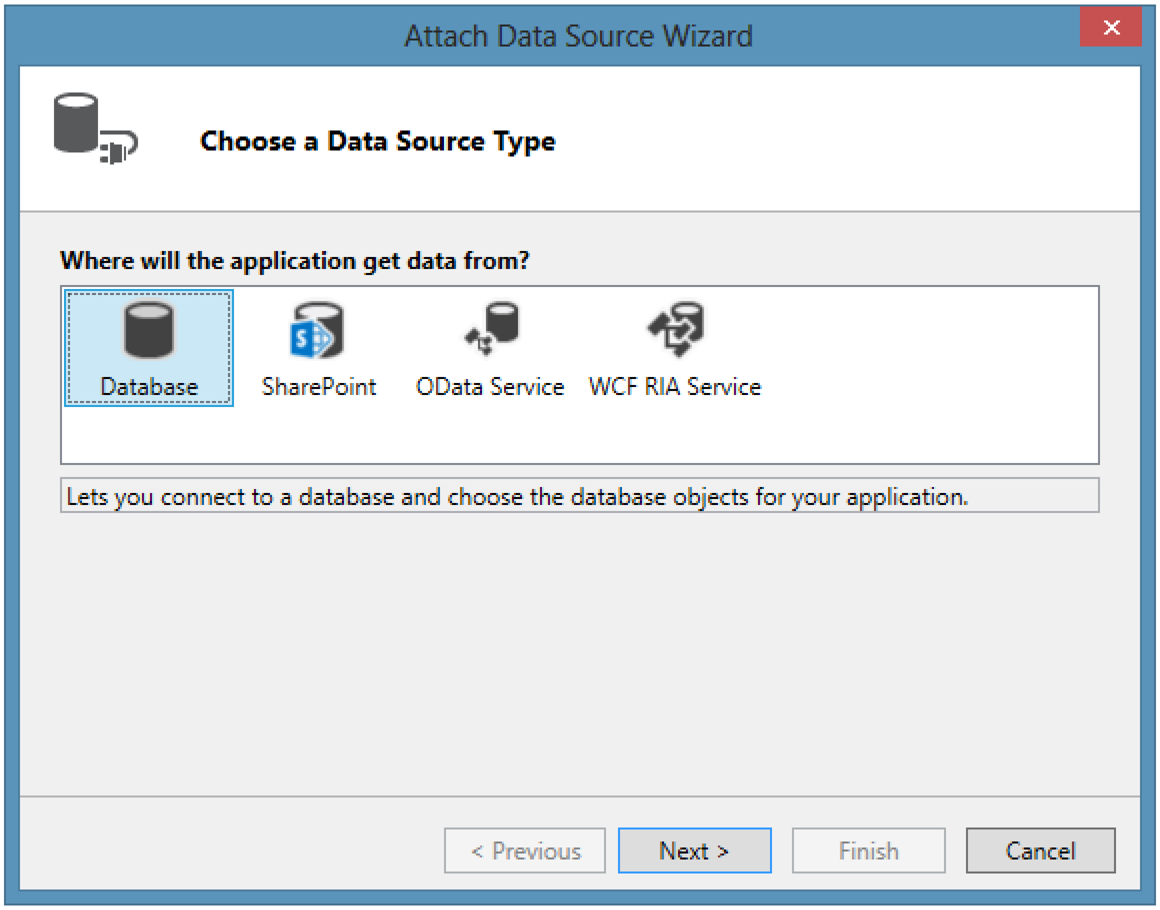


*Figure 21: The Intrinsic Data Source*

This data source will generate a related intrinsic database, that is a SQL Server or SQL Azure database, which will hold all entities that are specific to your application (**Intrinsic data source**).

LightSwitch also has options to connect to existing data sources (**Data source**). Selecting the option Add Data Source from the same context menu will open a wizard to help you connect to a variety of options.


Attach Data Source Wizard
Choose a Data Source Type
Where will the application get data from?
Database
SharePoint
OData Service
WCF RIA Service
< Previous
Next >
Finish
Cancel

Lets you connect to a database and choose the database objects for your application.

*Figure 22: Connecting to an Existing Data Source*

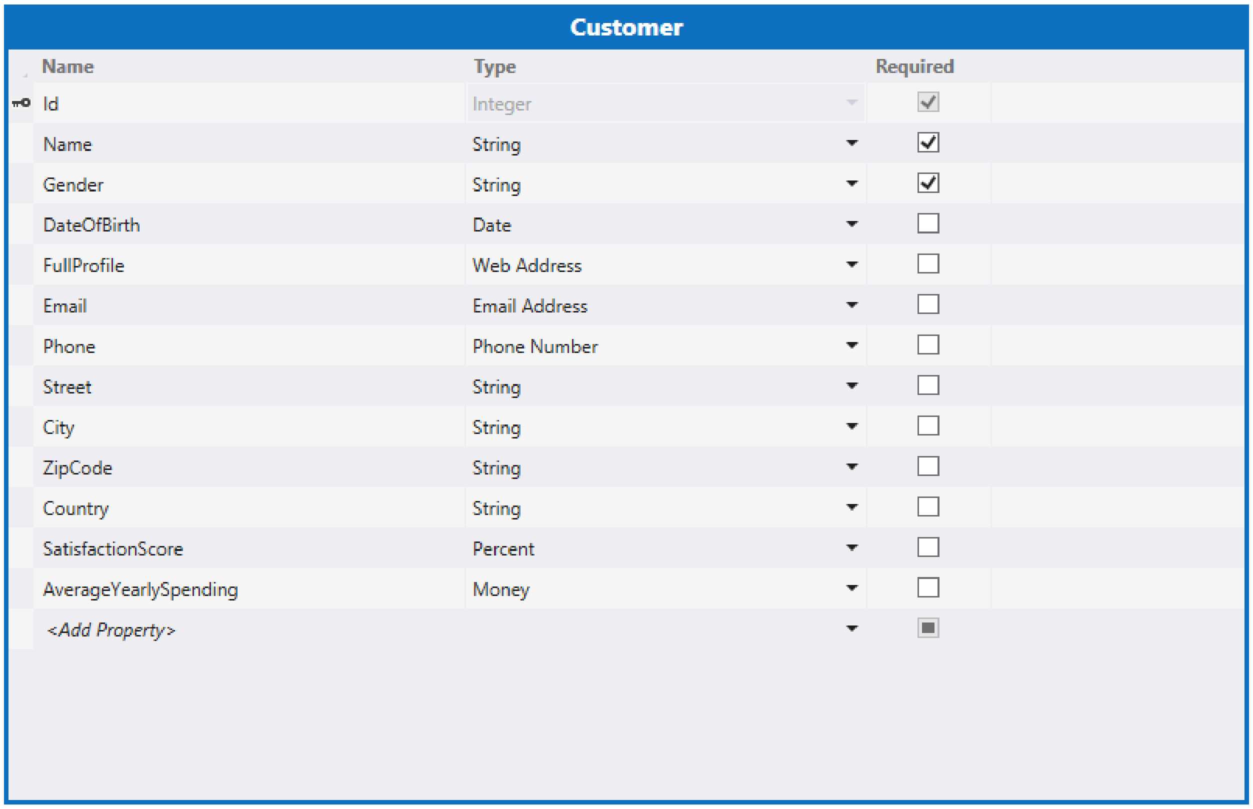
Database allows you to connect to any existing database.

SharePoint creates a connection to a SharePoint site so you can use any of the SharePoint lists in your application.

OData Service allows you to connect to any web service that is exposed via the Open Data Protocol (http://OData.org ). In its turn, every LightSwitch application uses JSON Light (an OData implementation) to expose the middle tier to LightSwitch or  third-party clients. This means you can use the OData Service connection to connect a new LightSwitch application to the entities in an already deployed LightSwitch application, boosting **reusability**.

A WCF RIA Service can be any class that inherits from the DomainService base class. This allows developers to quickly create an adapter in code between the LightSwitch application and anything they can connect to from code: CSV or XML files, SOAP services, Office documents, or whatever proprietary format they want.

The entity designer promotes object-oriented principles by making sure that you implement the logic where it belongs: encapsulated by the entity. Each field has a business type (**unique Business Types**). LightSwitch has almost 20 built-in business types, ranging from simple data types like Boolean or string to more complex types like E-mail or Money.



| Name | Type | Required |
|---|---|---|
| Id | Integer | ✓ |
| Name | String | ✓ |
| Gender | String | ✓ |
| DateOfBirth | Date | ☐ |
| FullProfile | Web Address | ☐ |
| Email | Email Address | ☐ |
| Phone | Phone Number | ☐ |
| Street | String | ☐ |
| City | String | ☐ |
| ZipCode | String | ☐ |
| Country | String | ☐ |
| SatisfactionScore | Percent | ☐ |
| AverageYearlySpending | Money | ☐ |
| *<Add Property>* | | ▣ |

*Figure 23: The Customer Entity in the Entity Designer*

Apart from proprietary built-in validation (for example: e-mail validation), each business type exposes some additional specialized properties in the properties window. Money, for example, will have both properties that are typical to the backing data field (decimal places to name one) and properties that are inherent to the business type (currency symbol for example).

If you connect to an existing data source, the previous type of properties, like the maximum length or size of the field, will be disabled for change.

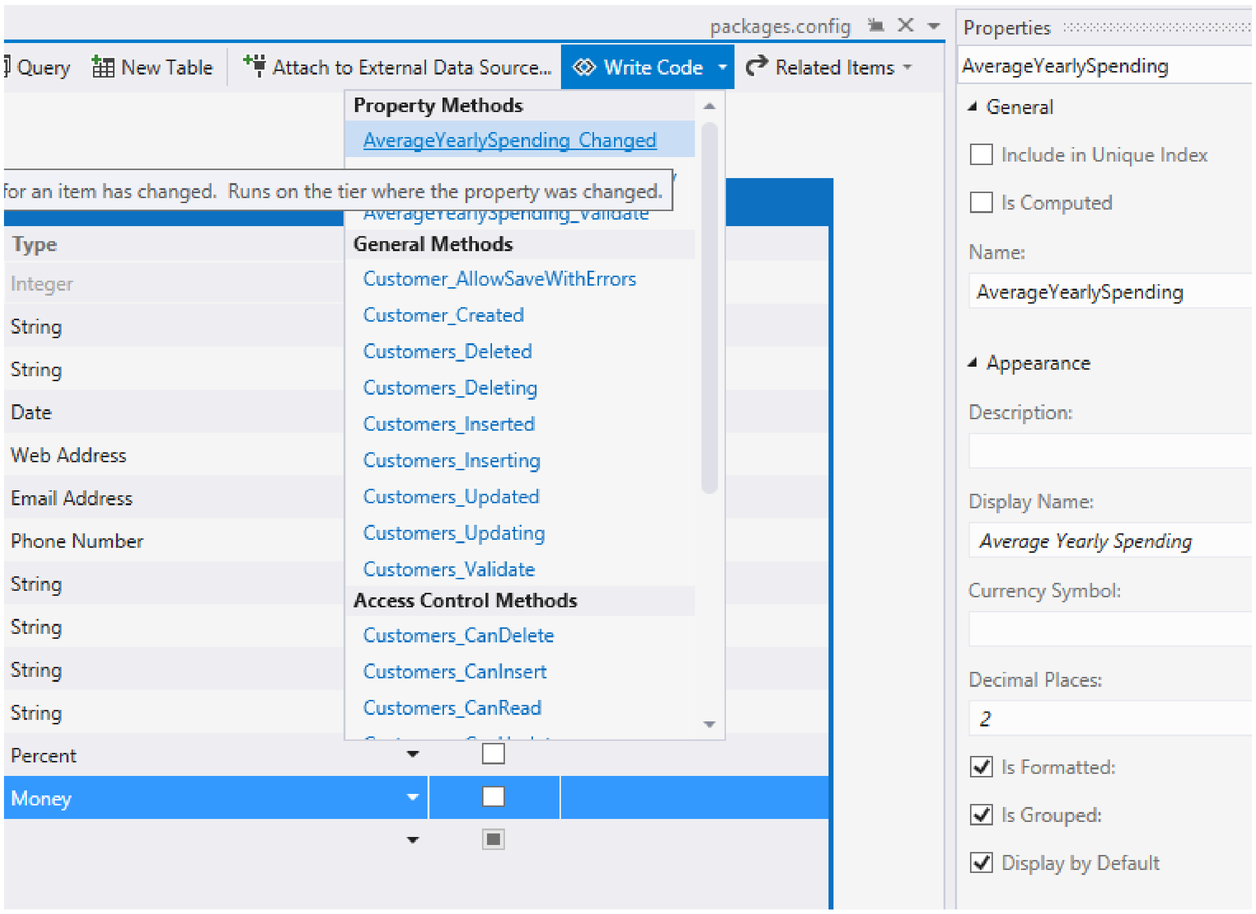


*Figure 24: The screen designer shows extended properties and code entry points.*

If you need more control to **encapsulate the business logic**, you can have a method or function stub generated by clicking Write Code at the top of the entity designer. Some methods will run server-side or client-side only, others run both on the server and on the client (for example: the 'created' method that executes when a new instance of a particular entity is created, useful for initialization purposes).

To avoid ambiguity, change the perspective at the bottom of the entity designer to indicate where you want your custom code to run.

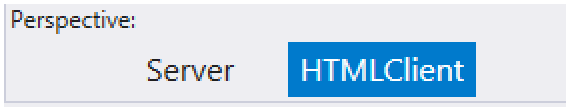
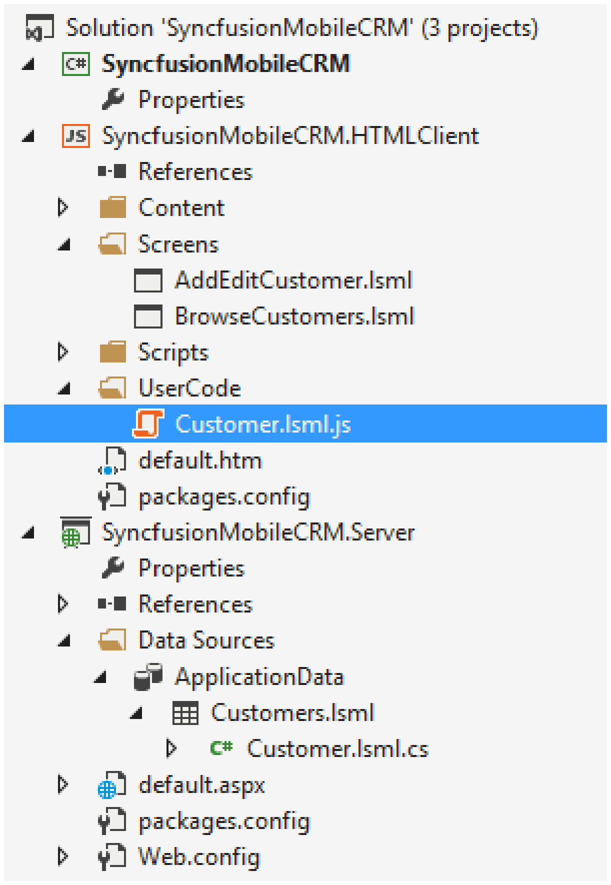


*Figure 25: Changing Perspective*

Depending on your perspective, a JavaScript or a C#-VB.NET class will automatically be added.



```
Solution 'SyncfusionMobileCRM' (3 projects)
◢ SyncfusionMobileCRM
      Properties
◢ SyncfusionMobileCRM.HTMLClient
      References
   ▷  Content
   ◢  Screens
         AddEditCustomer.lsml
         BrowseCustomers.lsml
   ▷  Scripts
   ◢  UserCode
         Customer.lsml.js
      default.htm
      packages.config
◢ SyncfusionMobileCRM.Server
      Properties
   ▷  References
   ◢  Data Sources
      ◢  ApplicationData
         ◢  Customers.lsml
            ▷  Customer.lsml.cs
   ▷  default.aspx
      packages.config
   ▷  Web.config
```

*Figure 26: Custom code stored in JavaScript or .NET code*

Although you cannot see this in the entity designer, each entity has automatically maintained columns for auditing (named CreatedBy, Created, ModifiedBy, and Modified) and concurrency checking (RowVersion). Unlike other platforms—such as WPF, ASP.NET, or Win Forms—you will not see any page with controls (which would be HTML or XAML), but rather a hierarchical representation of how the user interface will look. The LightSwitch libraries will use this abstraction to create the View layer at run time by dynamically adding new HTML elements to the root HTML element in the single HTML page. The View in the desktop clients is not generated at run time in the same way, but that technical implementation detail is beyond the scope of this e-book.

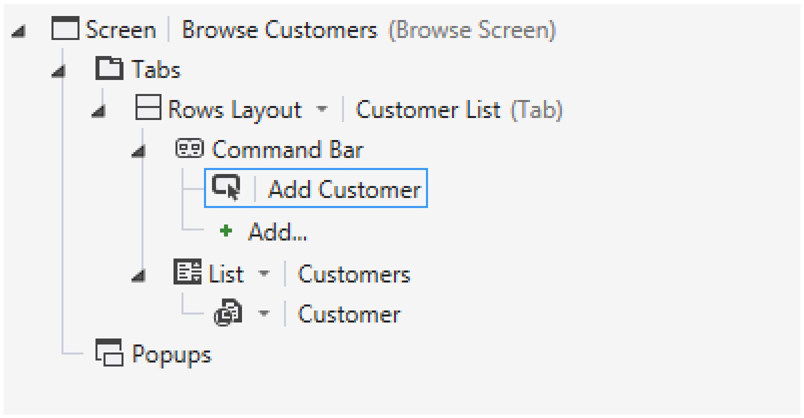
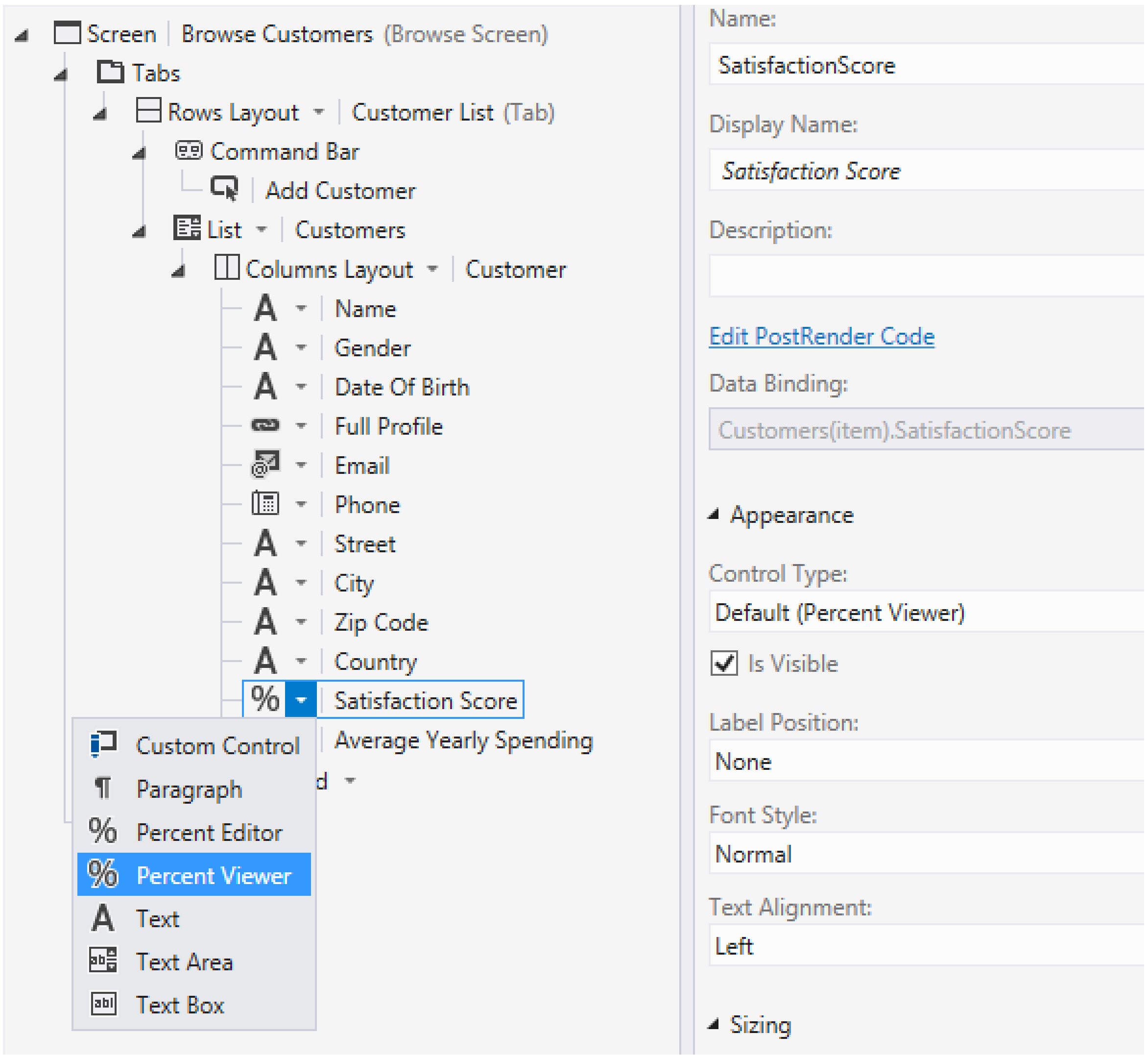

Screen | Browse Customers (Browse Screen)
Tabs
Rows Layout | Customer List (Tab)
Command Bar
Add Customer
Add...
List | Customers
Customer
Popups

*Figure 27: The View*

Each node in this visual tree is called ContentItem. A ContentItem is data-bound to a particular part of the ViewModel and provides additional information about the way it will be visualized to the user. Each ContentItem can be shown using the appropriate UI control based on the backing business type. For instance, percentage values can be shown via the Percent Viewer or Percent Editor controls. To choose a different control, click the dropdown arrow next to the node icon.

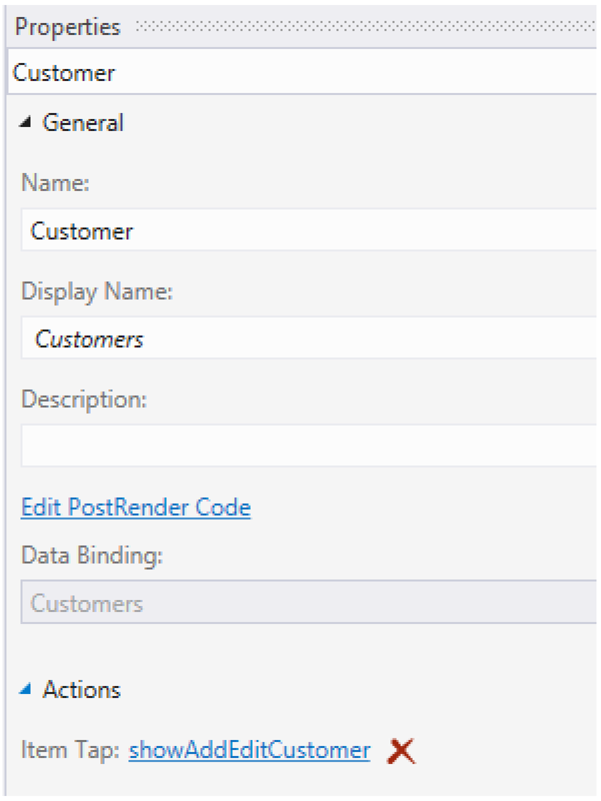

Screen | Browse Customers (Browse Screen)
Tabs
Rows Layout | Customer List (Tab)
Command Bar
Add Customer
List | Customers
Columns Layout | Customer
Name
Gender
Date Of Birth
Full Profile
Email
Phone
Street
City
Zip Code
Country
Satisfaction Score
Average Yearly Spending

Name: SatisfactionScore
Display Name: *Satisfaction Score*
Description:
Edit PostRender Code
Data Binding: Customers(item).SatisfactionScore
Appearance
Control Type: Default (Percent Viewer)
☑ Is Visible
Label Position: None
Font Style: Normal
Text Alignment: Left
Sizing

*Figure 28: Available Controls and Properties for a ContentItem in the View*

Also, each ContentItem node exposes additional properties in the properties window to further control the layout, size, font, and other properties. This will of course further be enhanced by the active cascading style sheet (CSS) in a mobile application or the selected theme in a Silverlight Desktop application. You can also add custom code to override the rendering. LightSwitch also offers specialized layout controls to arrange, group, and organize elements in the user interface and the ability to create a custom control from scratch. All the tooling you'll ever need to enhance your application with **typography, colors, custom layouts and specialized controls** is available, and we'll cover as much as possible in later chapters.

Besides looks, some properties allow you to set up actions or write custom code to respond to particular **touch gestures** with a single click.


Properties

Customer

General

Name:

Customer

Display Name:

Customers

Description:

Edit PostRender Code

Data Binding:

Customers

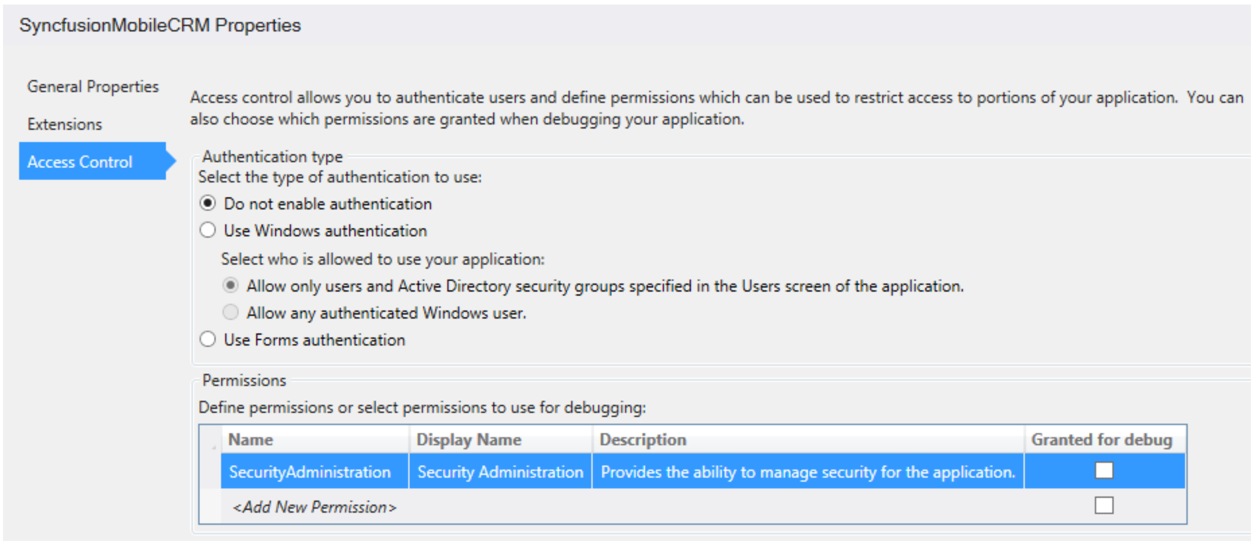
Actions

Item Tap: showAddEditCustomer

*Figure 29: Touch Gesture Actions like Item Tap*

Notice how there are no options, properties, or buttons in the screen designer to help create an **adaptive design,** as this is provided out-of-the-box, 100% effort-free.

After designing the application, it's time to select an **authentication** model. Select the LightSwitch application in Solution Explorer and open the Properties. The Access Control tab will allow you to select an authentication model that fits the existing IT infrastructure in the company with the click of a radio button.


SyncfusionMobileCRM Properties

Access control allows you to authenticate users and define permissions which can be used to restrict access to portions of your application. You can also choose which permissions are granted when debugging your application.

Authentication type

Select the type of authentication to use:

● Do not enable authentication

○ Use Windows authentication

Select who is allowed to use your application:

● Allow only users and Active Directory security groups specified in the Users screen of the application.

○ Allow any authenticated Windows user.

○ Use Forms authentication

Permissions

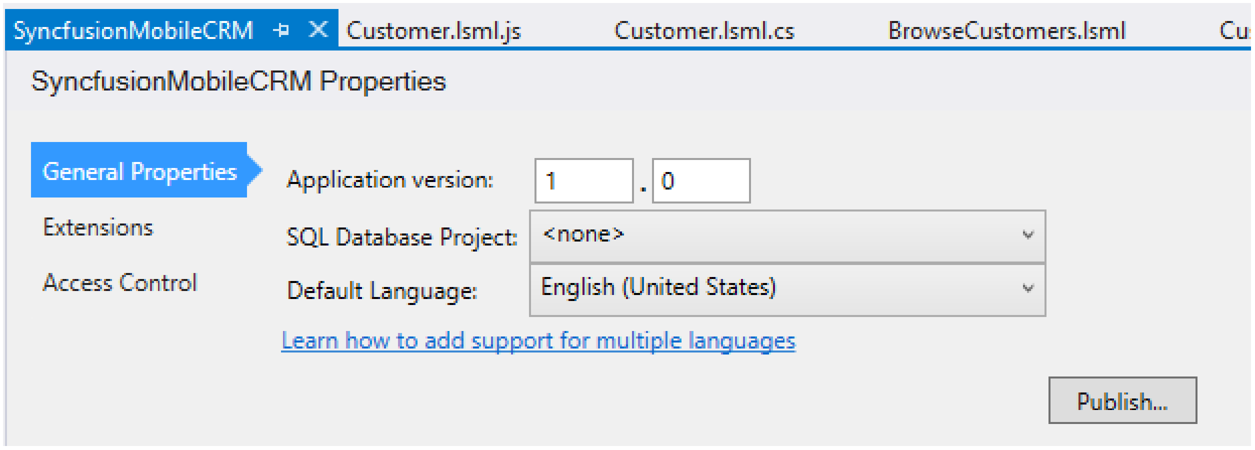
Define permissions or select permissions to use for debugging:

| Name | Display Name | Description | Granted for debug |
|---|---|---|---|
| SecurityAdministration | Security Administration | Provides the ability to manage security for the application. | ☐ |
| <Add New Permission> | | | ☐ |

*Figure 30: The Access Control Tab*

From the same tab, developers can add custom Permissions to the application. These permissions can then be checked in code to allow or restrict the end user to execute particular commands or open particular screens, but also for vertical or even horizontal data security like editing particular fields on an entity or even hiding particular rows from the database. An administrator (anyone with the SecurityAdministration Permission) can then group the end users into User Groups (or reuse the Active Directory security groups in the case of Windows Authentication), and assign a subset of these Permissions to each group. This allows for **authorization** to be as fine-grained as the business requires it to be.

From the General Properties tab, you can pick the application's language from 41 supported languages.



SyncfusionMobileCRM Properties

Application version: 1 . 0

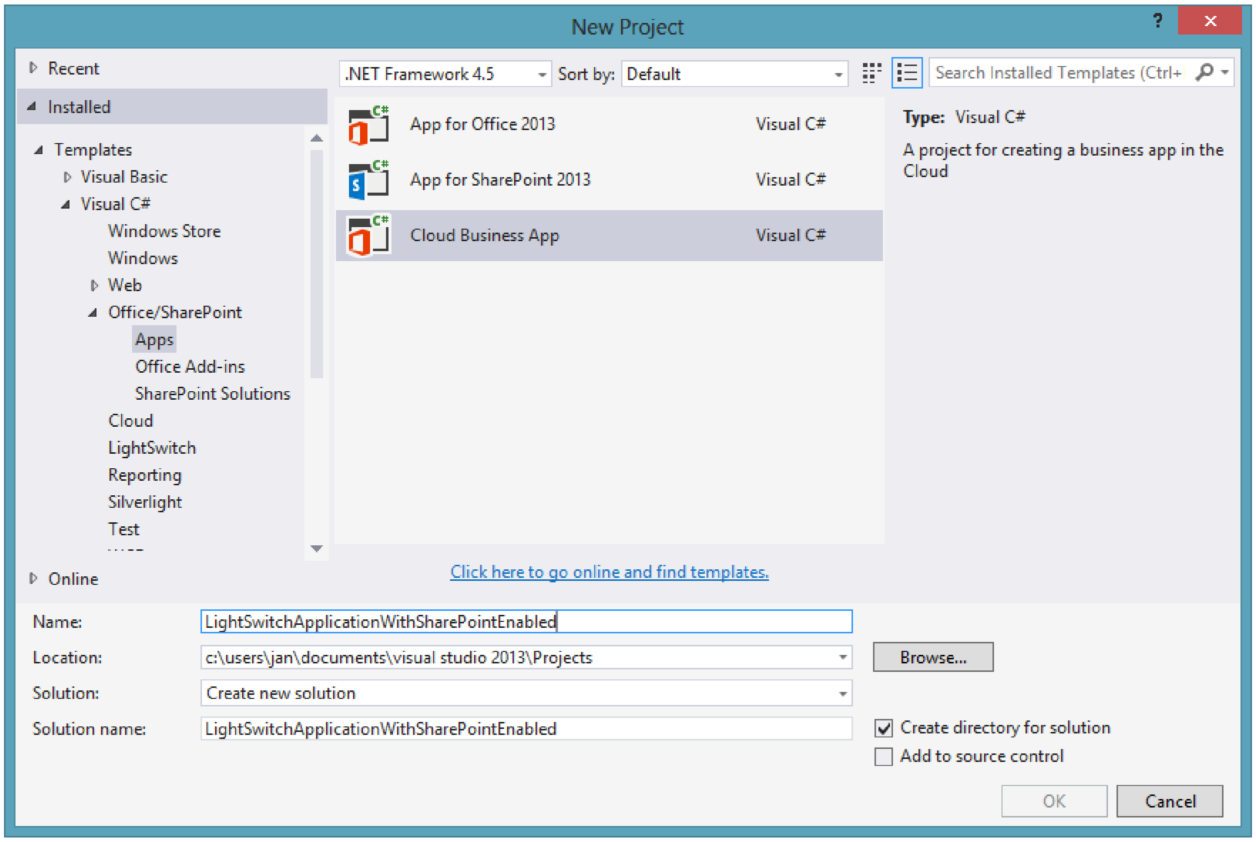
SQL Database Project: <none>

Default Language: English (United States)

*Figure 31: The General Properties of a LightSwitch Application*

A single application can also support multiple languages simultaneously. This isn't covered in this book, but you can find more information at http://msdn.microsoft.com/en-us/library/vstudio/xx130603.aspx.

The number of available tabs will differ depending on your type of LightSwitch application. We set out on a mission to create a mobile business app, but if you wanted to use LightSwitch to create Cloud Business Apps as well, just start your journey with the Cloud Business App project template, found under the Office/SharePoint node.



Type: Visual C#

A project for creating a business app in the Cloud

Name: LightSwitchApplicationWithSharePointEnabled

Location: c:\users\jan\documents\visual studio 2013\Projects

Solution: Create new solution

Solution name: LightSwitchApplicationWithSharePointEnabled

☑ Create directory for solution

☐ Add to source control

*Figure 32: Cloud Business App Project Template*

This project template will create a LightSwitch application that is pre-configured to be deployed to a SharePoint 2013 server, either on premises or in the Cloud, like an Office 365 site. SharePoint development is outside the scope of this e-book, but follow this link if you would like to learn more on the subject: http://msdn.microsoft.com/en-us/library/vstudio/jj969620.aspx.

Returning to the tabs that are available for your current project, let’s look at the Extensions tab. LightSwitch has an SDK that you can download from http://visualstudiogallery.msdn.microsoft.com/2381b03b-5c71-4847-a58e-5cfa48d418dc. With this SDK, you can create additional, reusable adapters for different Data Sources, new Business Types of Controls, Screen templates, and even company-specific shells and themes. Note that at the time of writing, these four extensions were limited in use for the desktop client only. These extensions would then be packaged in a VSIX installer. Once you have installed such a VSIX, you can activate it per LightSwitch project in this tab.

Some major component vendors, including Syncfusion, have already wrapped all of their Silverlight controls in reusable LightSwitch controls. If you are interested, check out Essential Studio for LightSwitch at http://www.syncfusion.com/products/lightswitch and see how easy it is to combine the rapid application development power of LightSwitch with the innovation of Syncfusion.

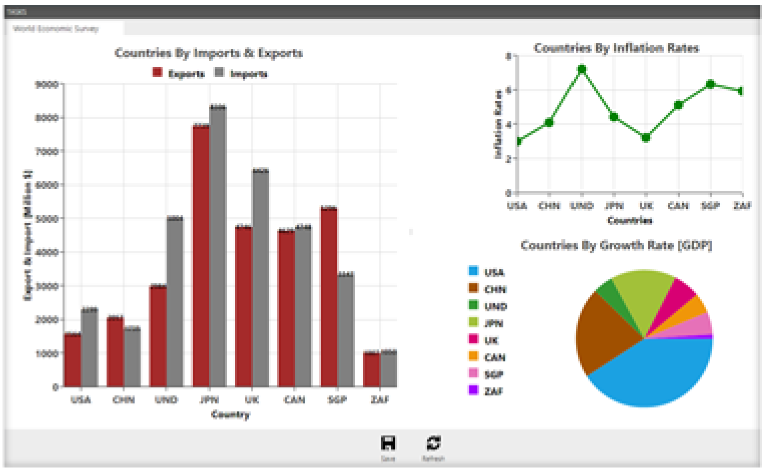

Countries By Imports & Exports
Exports
Imports
Export & Import (Million $)
Country
Countries By Inflation Rates
Inflation Rates
Countries
Countries By Growth Rate [GDP]
USA
CHN
UND
JPN
UK
CAN
SGP
ZAF
Save
Refresh

*Figure 33: Sample SilverLight Application with LightSwitch Controls*

Remember that these VSIX files have to be installed on your build server as well or else it will fail to compile your LightSwitch project based on the source code it just pulled from **source control**. LightSwitch uses editors in the IDE, but behind the scenes a LightSwitch solution contains normal projects that can be checked in or pushed to any source control, just like any other project. Additional guidelines and best practices on working on a LightSwitch project with multiple developers at the same time can be found at http://blogs.msdn.com/b/lightswitch/archive/2013/07/09/team-development-series-introduction-peter-hauge.aspx.

Before moving on, here's additional information about the elements in the General tab that I haven't mentioned yet: Version Number, Publish, and SQL Database Project.

LightSwitch uses semantic versioning. It's the responsibility of the developer to keep major and minor version numbers up to date. The patch number will be automatically increased each time you click the publish button.

Clicking the publish button will launch the publishing wizard. This process will be covered in more detail later. For now, I'll rest my case on LightSwitch being awesome tooling for LOB application development; I'll roll up my sleeves and  I'll dive into the last of the properties on the General Properties tab that I haven't mentioned yet: the SQL Database Project.

# Chapter 3 Introduction to SQL Database Projects

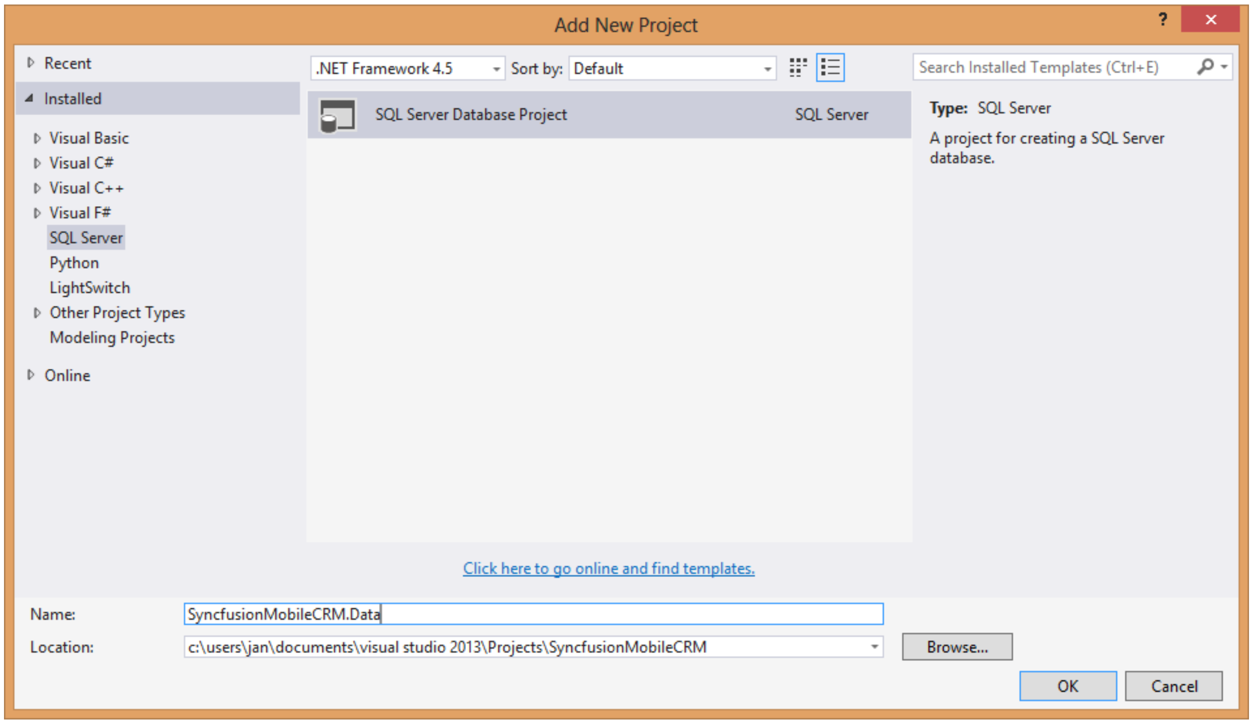
One of my favorite new LightSwitch features is the ability to add a database project to the solution to help shape the intrinsic database. This recent addition is in line with the LightSwitch vision: use simple editors to do the bulk of the work quickly while still retaining the ability to change even the tiniest technical detail on the lowest level.

This database project is intended to have a place to create stored procedures that can handle heavy loads of work or to add default or sample data or configuration. The LightSwitch project will keep the data model of this database project in sync with the entity designer (and execute any post-deployment scripts) each time you make a layout change in the entity designer.

This synchronization works in one direction only. The entity designer is considered the authority in regards to data models, and you should not use the database project to fundamentally alter the layout of the tables that will be generated based on your work in the entity designer.

SDET Chris Rummel explains this feature in detail at http://blogs.msdn.com/b/lightswitch/archive/2013/07/03/intrinsic-database-management-with-database-projects-chris-rummel.aspx.

For this e-book, we will focus on the essentials of SQL database projects. To begin, right-click on the solution and select Add New Project.

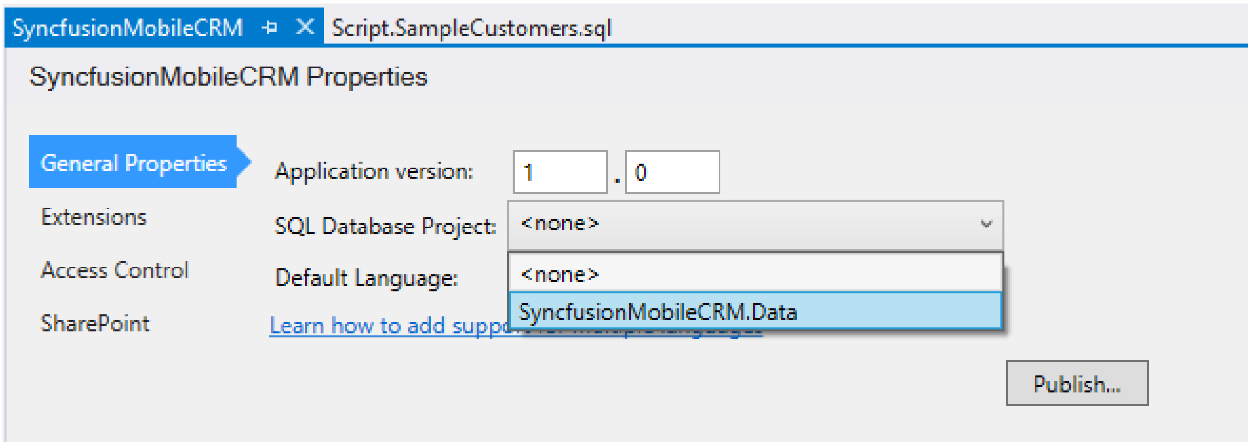


Name: SyncfusionMobileCRM.Data

Location: c:\users\jan\documents\visual studio 2013\Projects\SyncfusionMobileCRM

*Figure 34: Adding a New SQL Server Database Project*

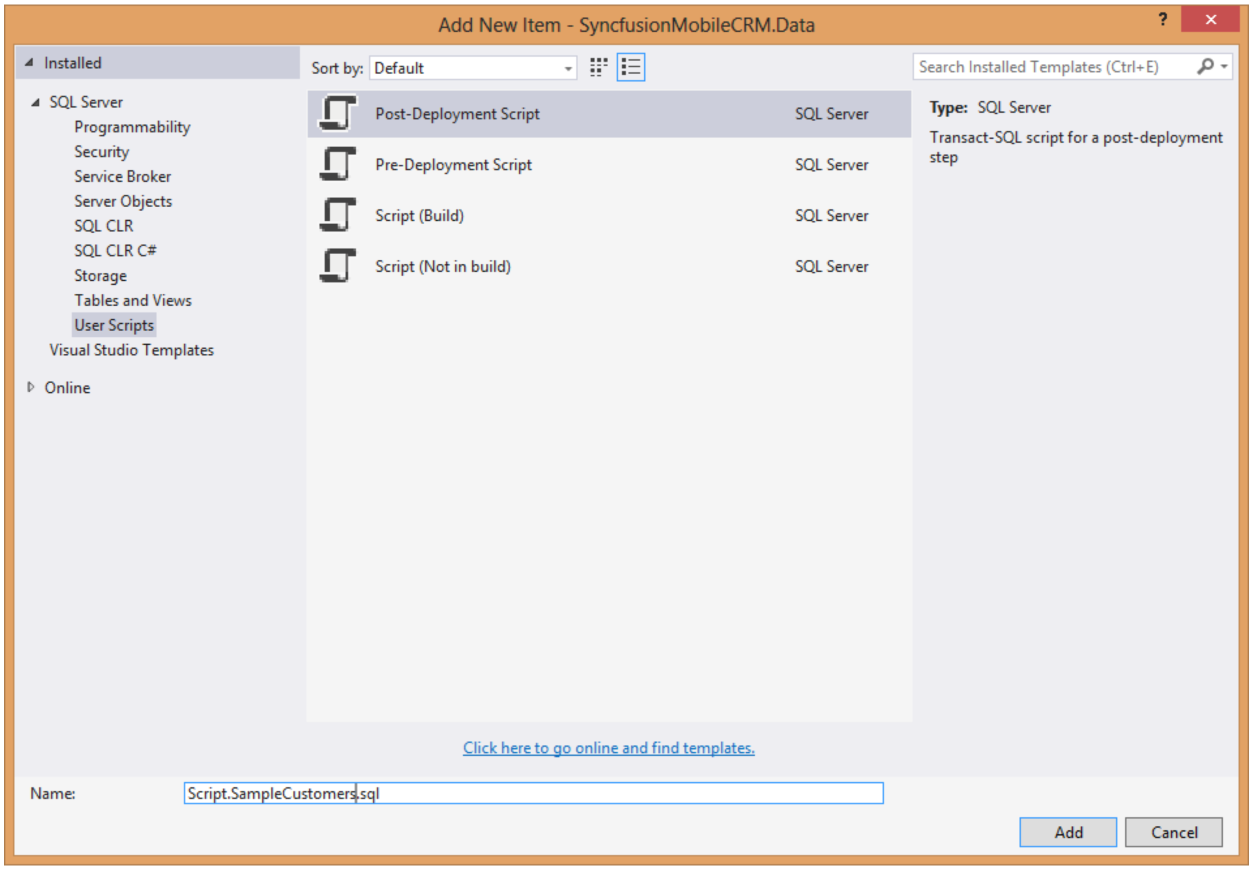
Once the project has been added to the solution, instruct the LightSwitch project to keep it in sync with the other projects.  To do this, right-click on the LightSwitch project and select Properties. From the General Properties tab, set the SQL Database Project to the project you have just created.



Application version: 1 . 0
SQL Database Project: <none>
<none>
SyncfusionMobileCRM.Data
Default Language:
Learn how to add support for multiple languages

Figure 35: Linking a Database Project

Now that the basic infrastructure is in place, add a new Post-Deployment Script to the database project.



| | |
|---|---|
| Post-Deployment Script | SQL Server |
| Pre-Deployment Script | SQL Server |
| Script (Build) | SQL Server |
| Script (Not in build) | SQL Server |

**Type:** SQL Server
Transact-SQL script for a post-deployment step

Name: Script.SampleCustomers.sql

Figure 36: Adding a Post-Deployment Script

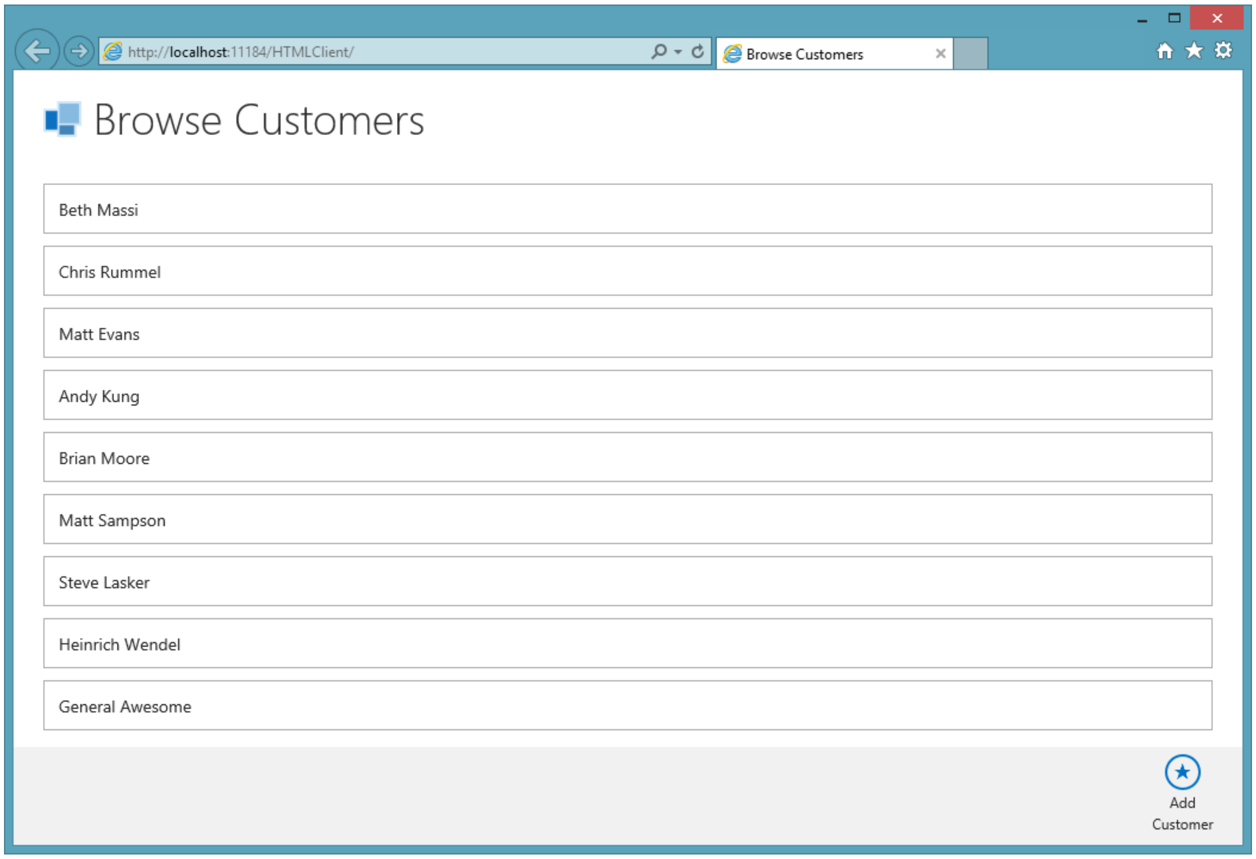
This post-deployment script will be executed each time the database is deployed. This includes when you publish the application for the first time, when you publish an update, and every time you make a change to the layout of your entities. I've updated Chris's SQL script to match our model.

```
SET IDENTITY_INSERT [dbo].[Customers] ON;


MERGE INTO [dbo].[Customers] AS Target

USING (VALUES

  (1, 'Beth Massi', 'F', 'Burg 1', '8000', 'Brugge', 'Belgium', 1.00),

  (2, 'Chris Rummel', 'M', 'Burg 1', '8000', 'Brugge', 'Belgium', 0.90 ),

  (3, 'Matt Evans', 'M', 'Burg 1', '8000', 'Brugge', 'Belgium', 0.80 ),

  (4, 'Andy Kung', 'M', 'Burg 1', '8000', 'Brugge', 'Belgium', 0.70 ),

  (5, 'Brian Moore', 'M', 'Burg 1', '8000', 'Brugge', 'Belgium', 0.60 ),

  (6, 'Matt Sampson', 'F', 'Burg 1', '8000', 'Brugge', 'Belgium', 0.50 ),

  (7, 'Steve Lasker', 'M', 'Burg 1', '8000', 'Brugge', 'Belgium', 0.40 ),

  (8, 'Heinrich Wendel', 'M', 'Burg 1', '8000', 'Brugge', 'Belgium', 0.30 ),

  (9, 'General Awesome', 'F', 'Burg 1', '8000', 'Brugge', 'Belgium', 0.00)

)

AS Source(Id, Name, gender, Street,  ZipCode, City, Country,
SatisfactionScore)

ON Target.Id = Source.Id

-- update matched rows

WHEN MATCHED THEN

UPDATE SET Name = Source.Name, gender = Source.gender, Street =
Source.Street,

      ZipCode = Source.ZipCode,  City = Source.City, Country =
Source.Country,

      SatisfactionScore = Source.SatisfactionScore, DateOfBirth = NULL,
      FullProfile = NULL, Email = NULL, Phone = NULL, AverageYearlySpending =
NULL

-- insert new rows

WHEN NOT MATCHED BY TARGET THEN

INSERT (Id, Name, gender, Street,  ZipCode, City, Country, SatisfactionScore)

VALUES (Id, Name, gender, Street,  ZipCode, City, Country, SatisfactionScore)

-- delete rows that are in the target but not the source

WHEN NOT MATCHED BY SOURCE THEN

DELETE;



SET IDENTITY_INSERT [dbo].[Customers] OFF;



GO
```

*Code Listing 1*

The intent of the script is simple: given a hardcoded set of customers, update the rows in the database with corresponding IDs, insert the missing ones, and remove the excess data.  Any resemblance between the characters in this data and any persons, living or dead, is pure coincidence (5).

Rebuild and launch the application again to verify our test subjects are in place.



Browse Customers

Beth Massi
Chris Rummel
Matt Evans
Andy Kung
Brian Moore
Matt Sampson
Steve Lasker
Heinrich Wendel
General Awesome

*Figure 37: Rebuilt Application*

The app shows the customers as expected.

# Chapter 4 Customizing Application Appearance

Right out of the box, the application was fully functional and had some basic styling. In this chapter, I'll show you a couple of techniques to further refine the appearance. First you'll use the screen designer, eventually dive into some basic CSS, JavaScript, and JQuery, and finally completely brand the global application theme.

## From the screen designer: Layout and controls

Our first objective is to turn the flat list of customers into a more dynamic overview that offers more information than only the name. From the screen designer select the customers element, and from the dropdown menu next to the List control select Table.

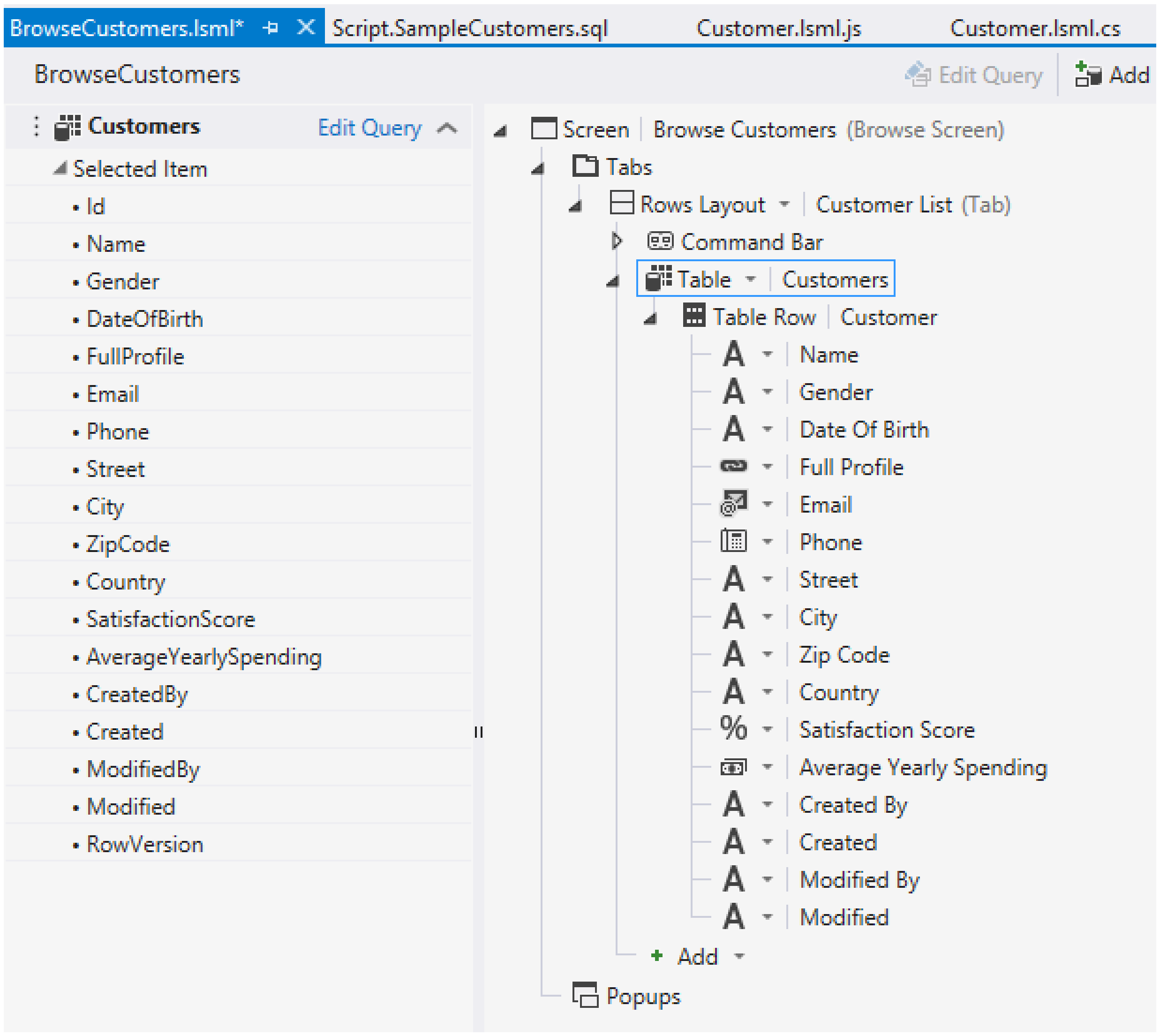
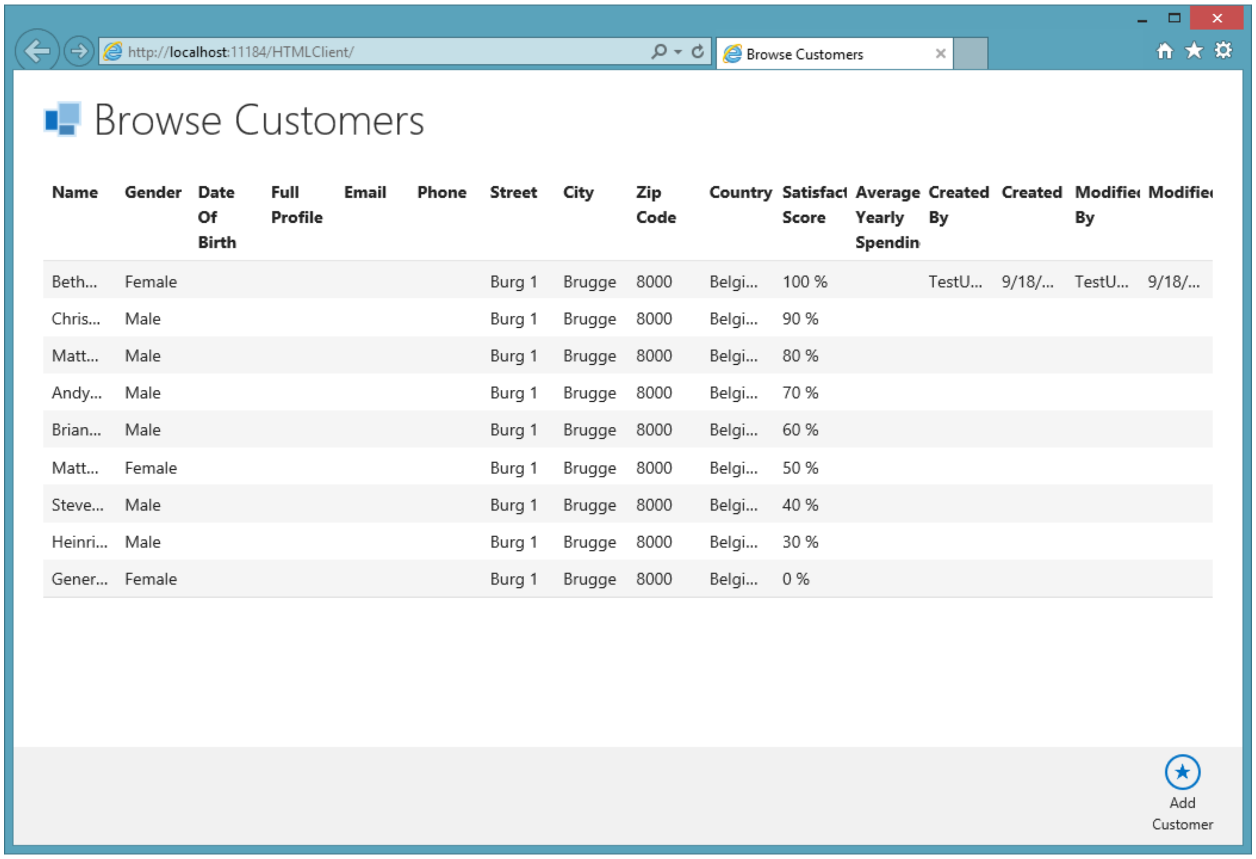


*Figure 38: Changing the Collection Control of the Customers ContentItem*

Save and then refresh the browser to see the change.



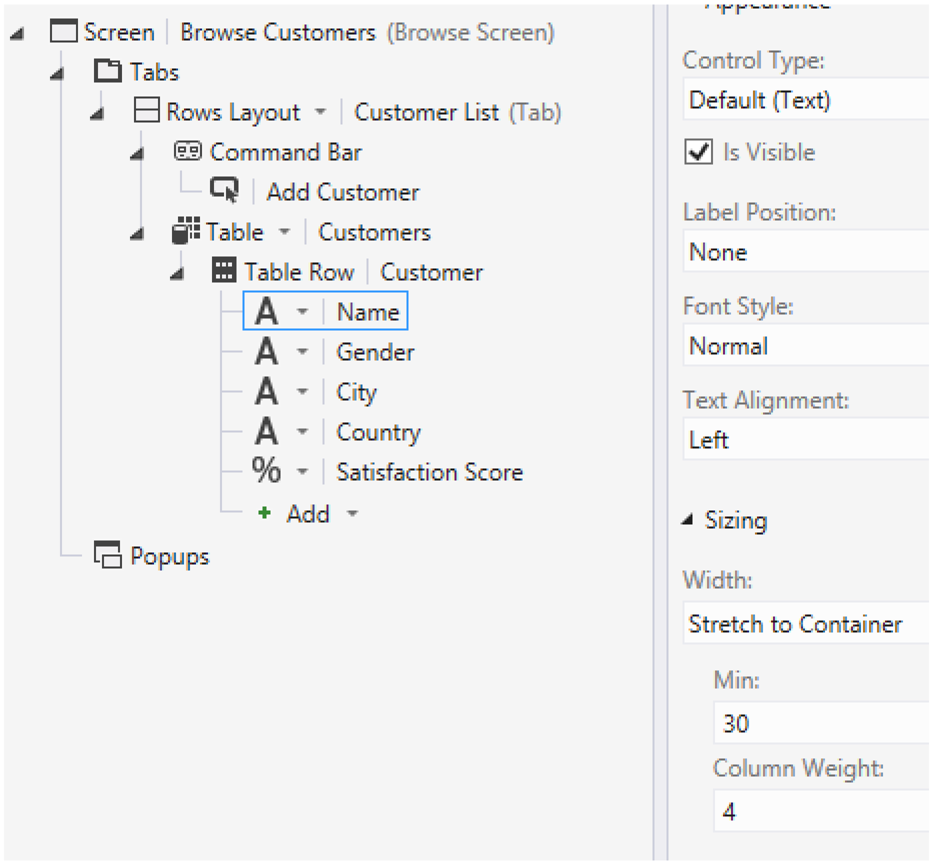
| Name | Gender | Date Of Birth | Full Profile | Email | Phone | Street | City | Zip Code | Country | Satisfact Score | Average Yearly Spendin | Created By | Created | Modifie By | Modifie |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Beth... | Female | | | | | Burg 1 | Brugge | 8000 | Belgi... | 100 % | | TestU... | 9/18/... | TestU... | 9/18/... |
| Chris... | Male | | | | | Burg 1 | Brugge | 8000 | Belgi... | 90 % | | | | | |
| Matt... | Male | | | | | Burg 1 | Brugge | 8000 | Belgi... | 80 % | | | | | |
| Andy... | Male | | | | | Burg 1 | Brugge | 8000 | Belgi... | 70 % | | | | | |
| Brian... | Male | | | | | Burg 1 | Brugge | 8000 | Belgi... | 60 % | | | | | |
| Matt... | Female | | | | | Burg 1 | Brugge | 8000 | Belgi... | 50 % | | | | | |
| Steve... | Male | | | | | Burg 1 | Brugge | 8000 | Belgi... | 40 % | | | | | |
| Heinri... | Male | | | | | Burg 1 | Brugge | 8000 | Belgi... | 30 % | | | | | |
| Gener... | Female | | | | | Burg 1 | Brugge | 8000 | Belgi... | 0 % | | | | | |

*Figure 39: Customers are now shown in a table.*

By default, each field in the customer entity will be shown as a column, including the automatically generated and maintained auditing columns (but excluding the RowVersion column used for concurrency checking on the server).

If you are wondering why the first record has a date in the auditing fields and the others don't, it is because in my case, the first row was created using the application. The contents of that same record were later overwritten by the SQL script directly in the database because the script matches on ID. The auditing columns are maintained by the middle tier (the OData service). This is why the rows that were created by our SQL post-deployment script have no auditing information.

Let's tidy this up a bit. Back in the screen designer, select any sub-node of the Table Row node and press delete on your keyboard or click the delete button in the command bar. Repeat until you have about five columns left.

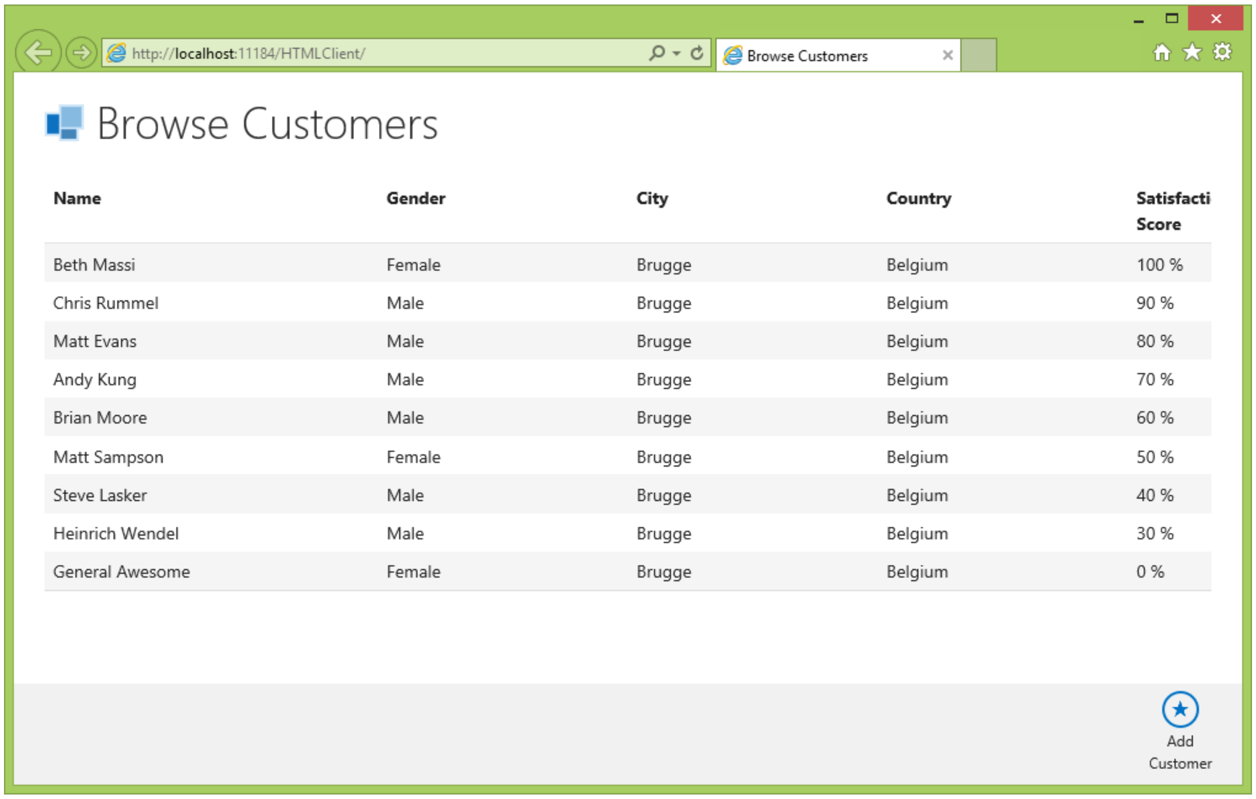

Screen | Browse Customers (Browse Screen)
Tabs
Rows Layout | Customer List (Tab)
Command Bar
Add Customer
Table | Customers
Table Row | Customer
Name
Gender
City
Country
Satisfaction Score
Add
Popups

Control Type: Default (Text)
☑ Is Visible
Label Position: None
Font Style: Normal
Text Alignment: Left
Sizing
Width: Stretch to Container
Min: 30
Column Weight: 4

*Figure 40: Removing Excess Columns*

Not every column needs the same amount of space to display the contents. Using the properties window, you can select some of the columns and give them a Fixed Width or have them resize automatically based on the available width by setting them to Stretch to Container. In the latter case, columns with a larger Column Weight will be given a larger proportion of the available width.

Again, saving and refreshing displays the updated layout.

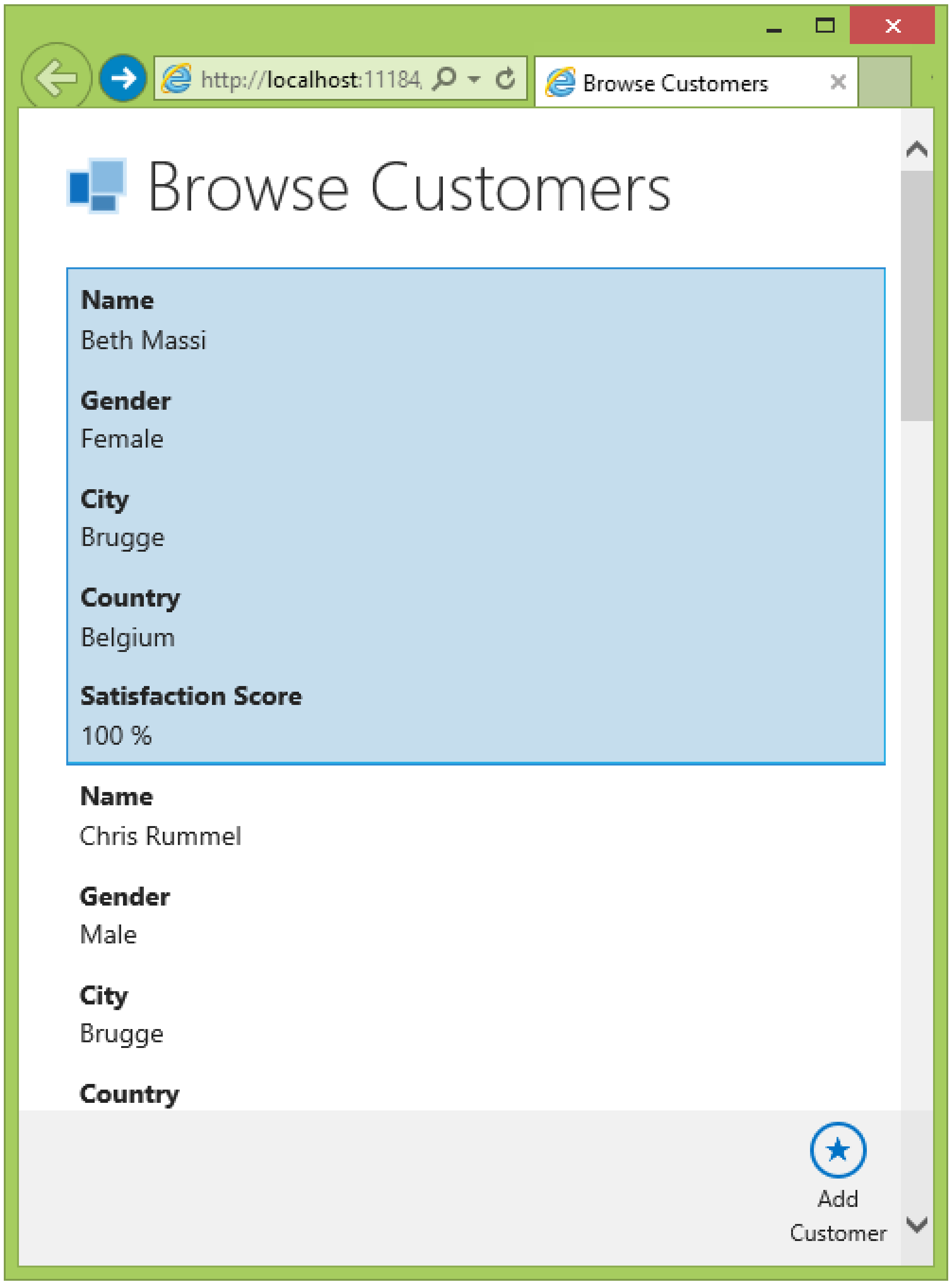

http://localhost:11184/HTMLClient/

Browse Customers

| Name | Gender | City | Country | Satisfacti Score |
|---|---|---|---|---|
| Beth Massi | Female | Brugge | Belgium | 100 % |
| Chris Rummel | Male | Brugge | Belgium | 90 % |
| Matt Evans | Male | Brugge | Belgium | 80 % |
| Andy Kung | Male | Brugge | Belgium | 70 % |
| Brian Moore | Male | Brugge | Belgium | 60 % |
| Matt Sampson | Female | Brugge | Belgium | 50 % |
| Steve Lasker | Male | Brugge | Belgium | 40 % |
| Heinrich Wendel | Male | Brugge | Belgium | 30 % |
| General Awesome | Female | Brugge | Belgium | 0 % |

*Figure 41: A Reformatted Customer Table*

The useful thing about displaying collections in a Table control is that the control has a unique way of adapting to smaller screen factors. If the screen width is smaller than the combined minimum width of each column (30 pixels by default), the screen will change orientation and display each column vertically instead of horizontally per row.



Browse Customers

**Name**
Beth Massi

**Gender**
Female

**City**
Brugge

**Country**
Belgium

**Satisfaction Score**
100 %

**Name**
Chris Rummel

**Gender**
Male

**City**
Brugge

**Country**

*Figure 42: Adaptive Design of the Table Control in Action*

Impressive, but let's give the third layout control a chance before we move on. Select the customers node in the screen designer, and from the dropdown menu replace Table with Tile List.

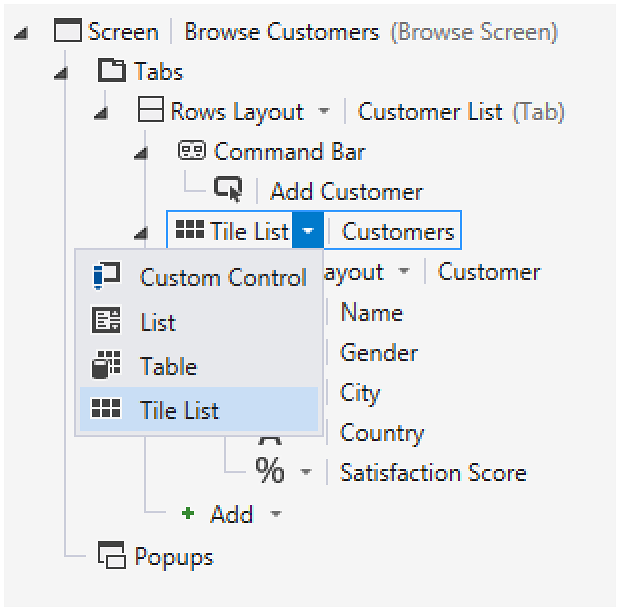


*Figure 43: Tile List Collection Control*

To prevent cramming too much information in each tile, I further removed all customer properties except for Name and Satisfaction Score. If you happened to have removed any of these columns, simply select the Rows Layout ContentItem that is bound to the customer, and click Add at the bottom.

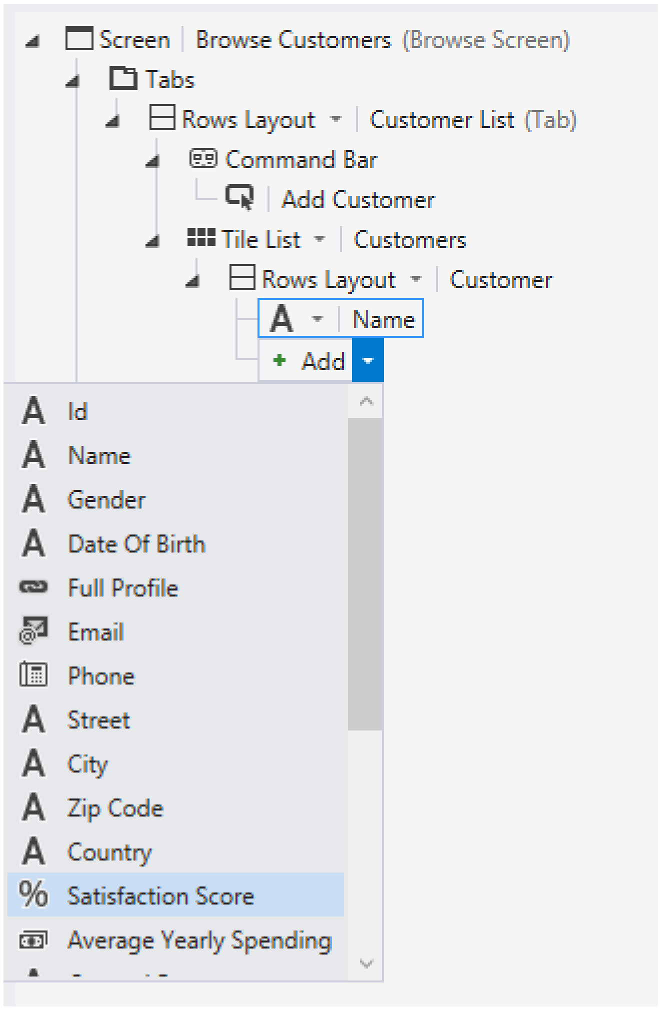
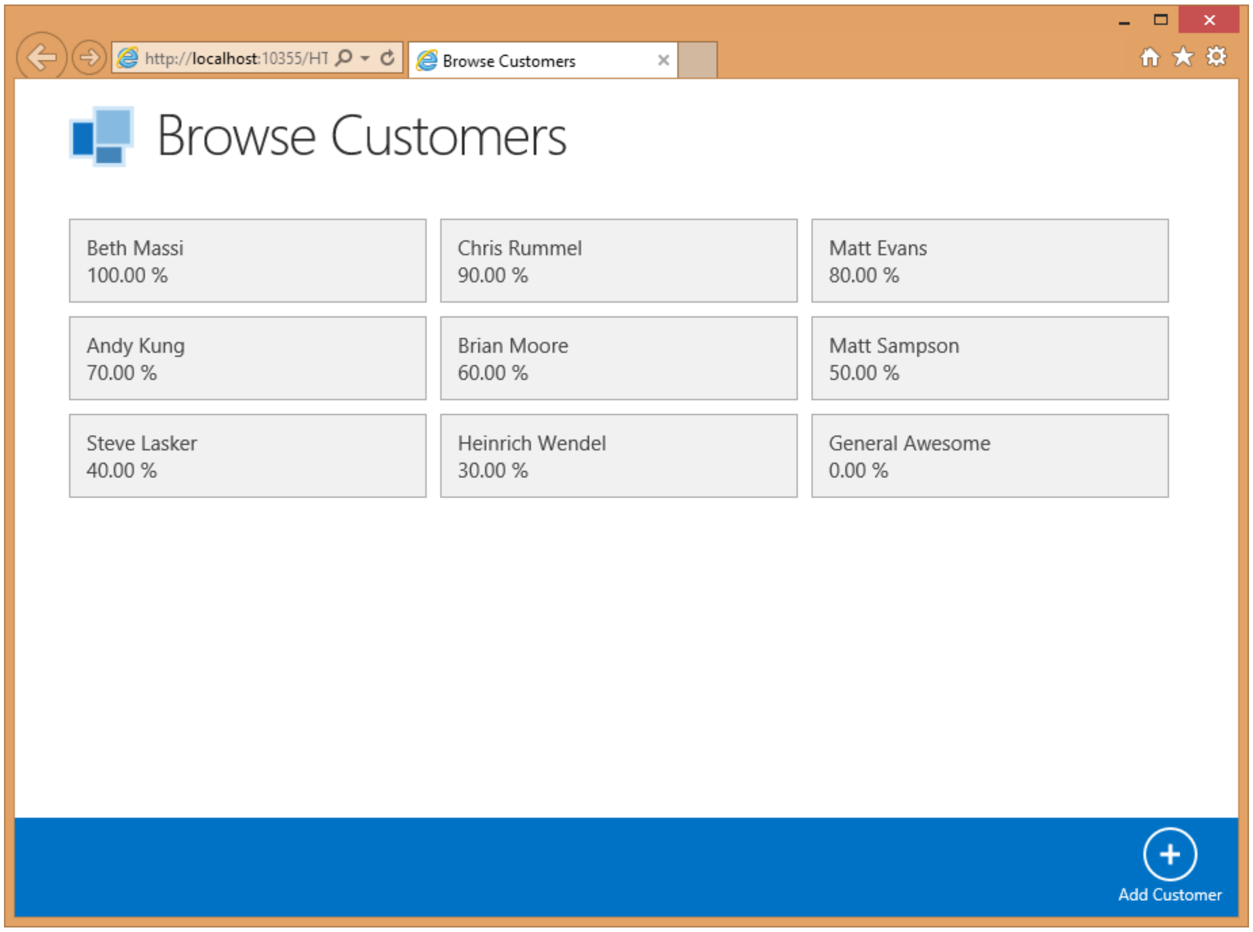


*Figure 44: Adding Fields Back to the View*

Saving and refreshing shows the Tile List in action.



| | | |
|---|---|---|
| Beth Massi 100.00 % | Chris Rummel 90.00 % | Matt Evans 80.00 % |
| Andy Kung 70.00 % | Brian Moore 60.00 % | Matt Sampson 50.00 % |
| Steve Lasker 40.00 % | Heinrich Wendel 30.00 % | General Awesome 0.00 % |

*Figure 45: The Tile List*

A Tile List control will also adapt to the available screen width like the Table control. The Tile List does this by reducing the number of columns. On smartphones, there would be only one column of Tiles, whereas tablets usually have a sufficiently large screen to display three or four columns.

## From the screen designer: Popups and filtering

By choosing the right combination of collection controls and layouts, you can make sure that the application shows as much useful information as possible, and only the useful information, for which your end users will be grateful. Let's explore some possibilities for filtering the data.

To filter the data, we need to change the query that is used to retrieve the data. Select the customer collection from the view model in the screen designer (the left-hand side) and select the Edit Query link.

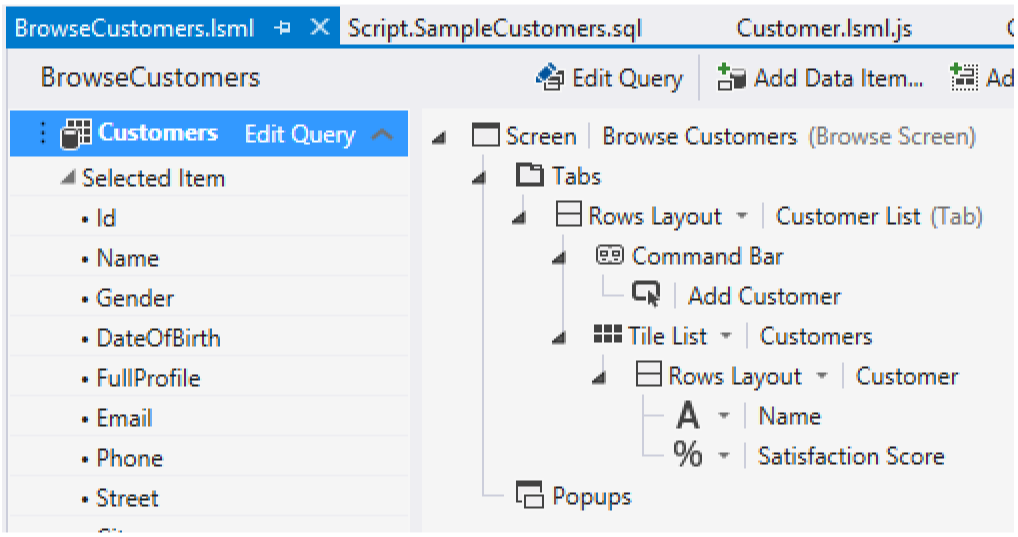


*Figure 46: You can edit each query on the view model.*

This will open up the collection in the query designer.

This designer makes it easy to sort the data or filter it by any combination of limitations on Literals (fixed values), other Properties (fields), parameters, or sometimes even smart business values (globals like “end of the current month”).

There are two ways in which the query designer can be used: on the server or on the client. To use it on the server, select any entity in the solution explorer and from the context menu select edit query. This creates a new query on the server, which will have a dedicated URI in the OData service, and can be reused between different screens.

Since we will not reuse this query, we will edit the query on the client. By doing so, the HTML client will append the correct OData commands when retrieving data from the server. Remember that in both cases, the actual filtering and sorting will be done on the server.

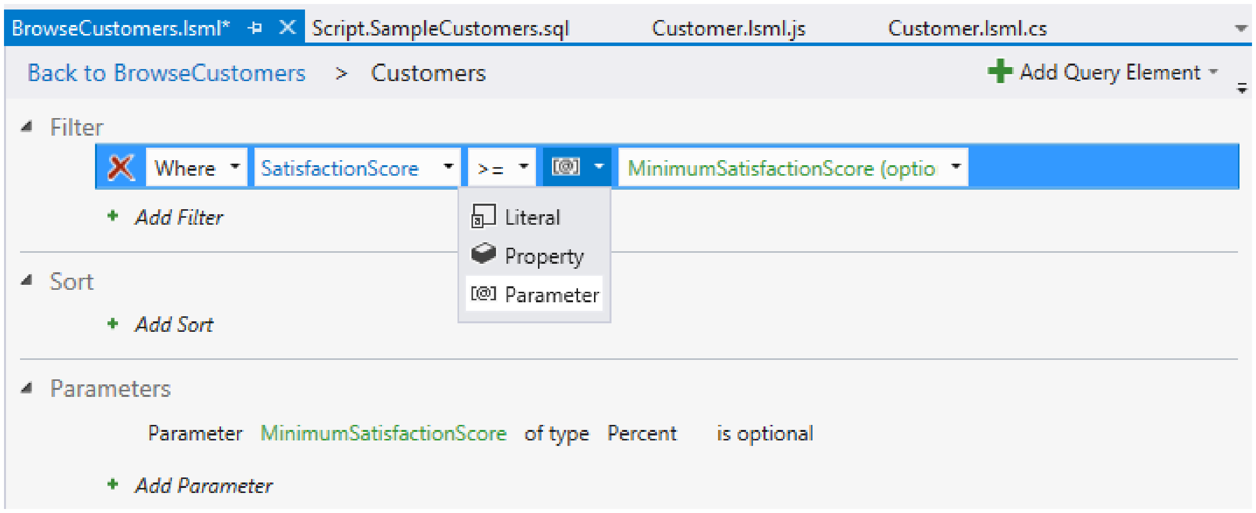


*Figure 47: The Query Designer*

Alter the query so it returns only those customers where the SatisfactionScore is higher than a particular value. This value will be passed as a parameter. I named mine MinimumSatisfactionScore. Before saving and navigating back to the screen designer by hitting the back button to the browseCustomers link at the top, make sure you select the newly created MinimumSatisfactionScore parameter in the query designer. From the properties window, select the is optional box.

Back in the screen designer, you'll notice that the customer collection on our view model now has the newly added query parameter called MinimumSatisfactionScore.

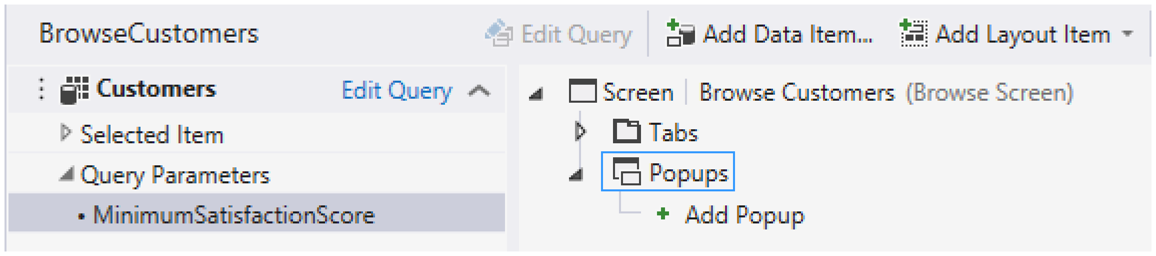


*Figure 48: The MinimumSatisfactionScore was added as a query parameter.*

From the view on the right-hand side, select the Popups node in the visual tree and click add to add a popup.

A new rows layout node will appear as the root node of your popup. Select it and, from the properties window, change the name to FilterPopup or something more meaningful than the default group.

Drag and drop the new query parameter MinimunSatisfactionScore from the view model onto the newly created popup in the view.

This dragand-drop operation will have two effects. First, a local screen property will be added to the view model (on the left-hand side) named MinimumSatisfactionScore. This screen property will be data-bound to the query parameter, so any changes to this property will change the parameter and cause the customers collection to be refreshed automatically. You can see that the local screen property is data-bound to the query parameter because of the arrow between the two. You can remove the ambiguity by renaming either, for example by selecting the MinimumSatisfactionScore screen property and, from the properties window, setting the name to HappinessFactor.

Additionally, a percentage editor has been added to the view, on the right-hand side, to show this local HappinessFactor screen property.

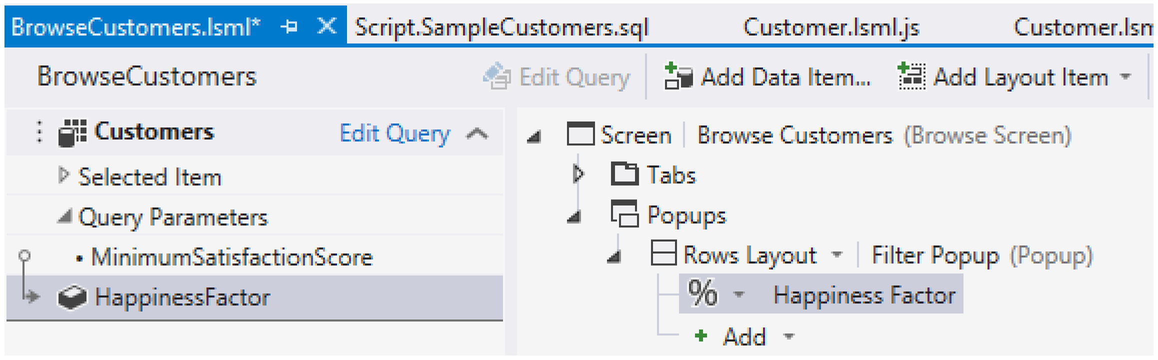


*Figure 49: State of the Screen Designer after Dragging the Query Parameter onto the Popup*

Now that the popup is in place, you need some kind of code that will open the popup at run time. The easiest way to set this up is by selecting the screen command bar and adding a new button. This will automatically open the add button wizard. By default, the navigation action showPopup will be selected, so just click OK to finalize the button.

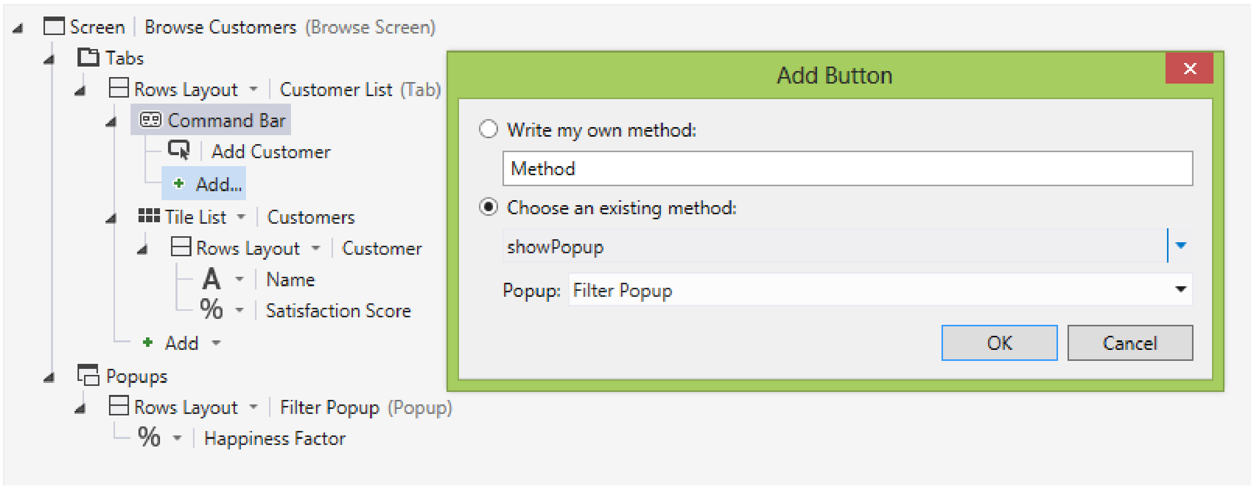
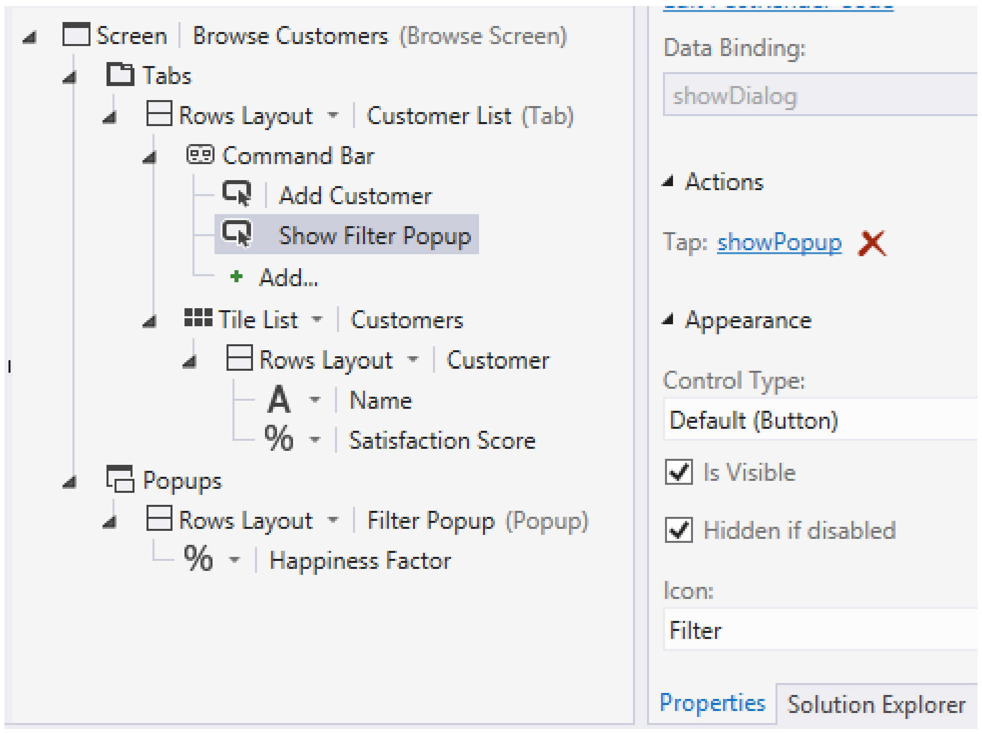


*Figure 50: Opening a Popup from the Add Button Wizard*

With the newly created button selected, look at the properties window and select any of the built-in icons as the Icon to use. I used the filter one for this example.


Screen | Browse Customers (Browse Screen)
Tabs
Rows Layout | Customer List (Tab)
Command Bar
Add Customer
Show Filter Popup
Add...
Tile List | Customers
Rows Layout | Customer
Name
Satisfaction Score
Popups
Rows Layout | Filter Popup (Popup)
Happiness Factor

Data Binding: showDialog

Actions
Tap: showPopup

Appearance
Control Type: Default (Button)
Is Visible ✓
Hidden if disabled ✓
Icon: Filter

Properties | Solution Explorer

*Figure 51: Selecting an Icon for the Newly Added Button*

Save your work and refresh your browser to see the result in action.

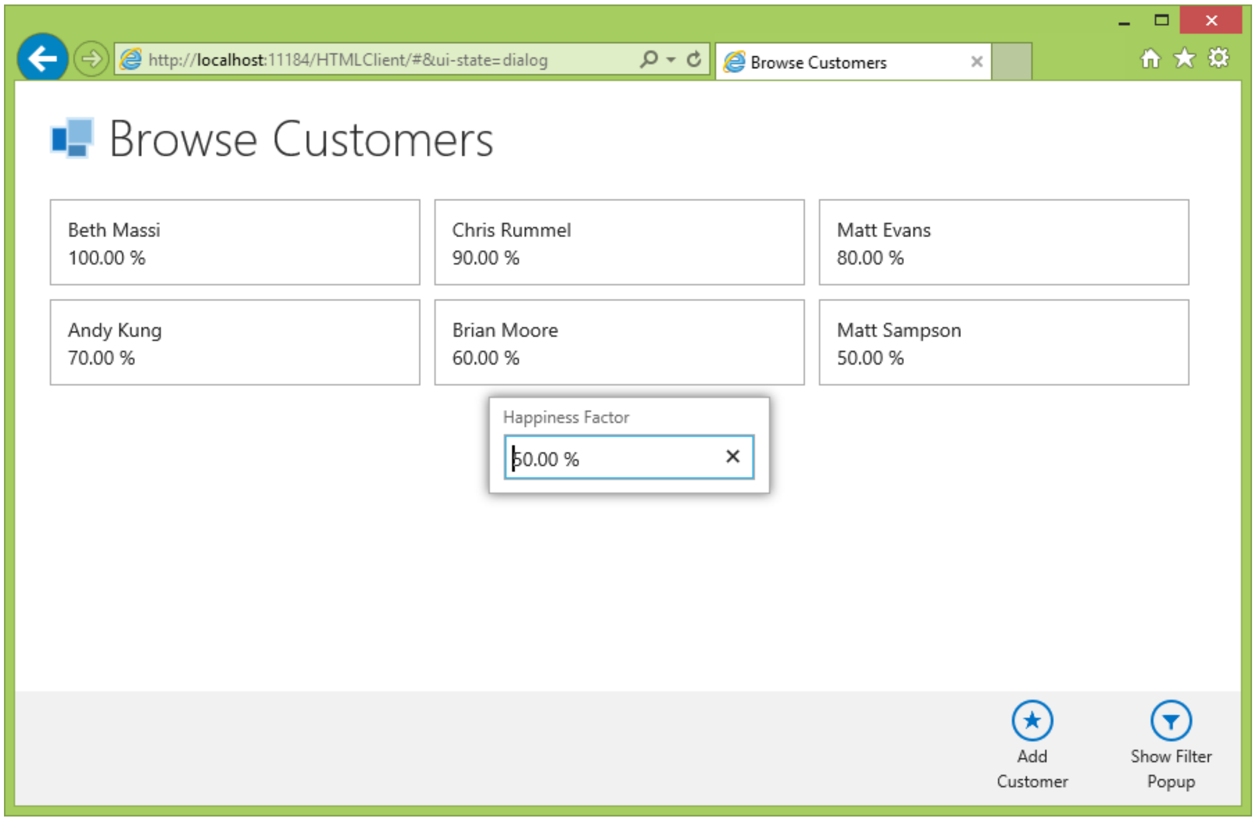


*Figure 52: The Filter Popup in Action*

The only thing that bothers me is forcing the end user to enter a numerical value. I would much rather present the end user with a choice list with some visual feedback instead.

At first it seems like this is not supported by the screen designer. The HappinessFactor ContentItem node in the screen's view only offers custom control (more about that later), percentage editor, text box, text area, and a couple of read-only options.

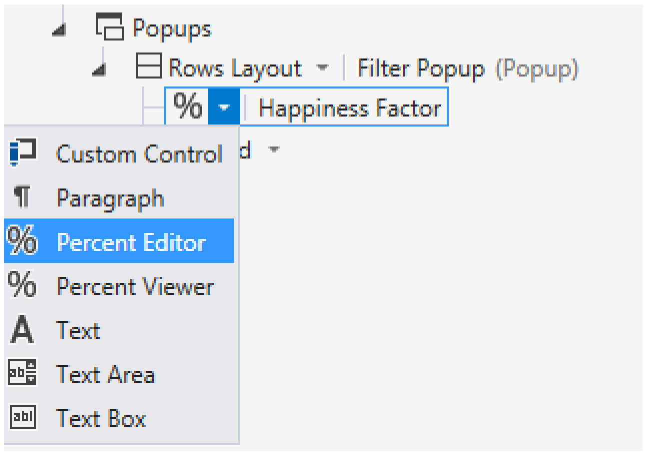
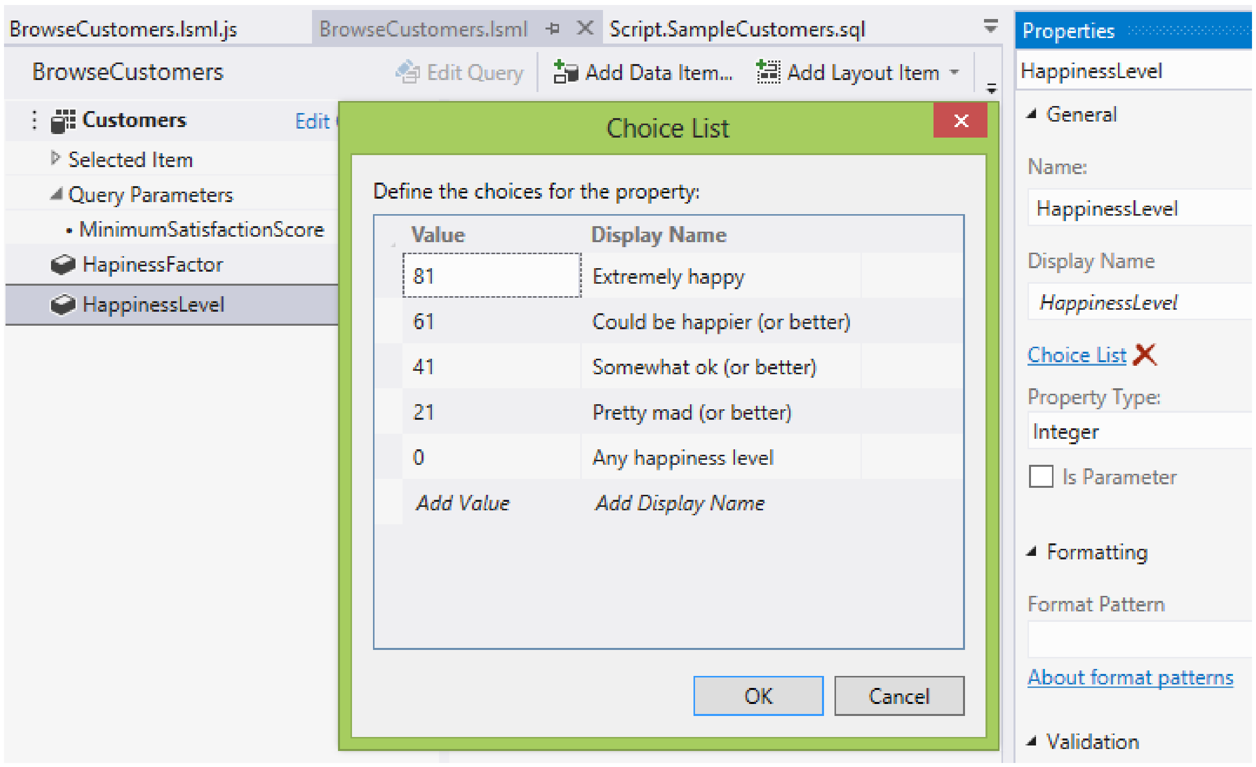


*Figure 53: Available Controls to Render the Happiness Factor*

## The power of local screen properties and data binding

Choice lists are only supported for string and integer properties or fields, so we'll need to add a workaround.

From the top of the screen designer, click the add data item button. The wizard that appears allows you to create a new Local Property of type integer, called HappinessLevel. With this newly created HappinessLevel selected in the View Model, click the Choice List link in the properties window. A dialog will appear where you can restrict this HappinessLevel property to a few predefined values, each with a descriptive display name.



Choice List

Define the choices for the property:

| Value | Display Name |
|---|---|
| 81 | Extremely happy |
| 61 | Could be happier (or better) |
| 41 | Somewhat ok (or better) |
| 21 | Pretty mad (or better) |
| 0 | Any happiness level |
| *Add Value* | *Add Display Name* |

HappinessLevel

Name: HappinessLevel

Display Name: *HappinessLevel*

Choice List

Property Type: Integer

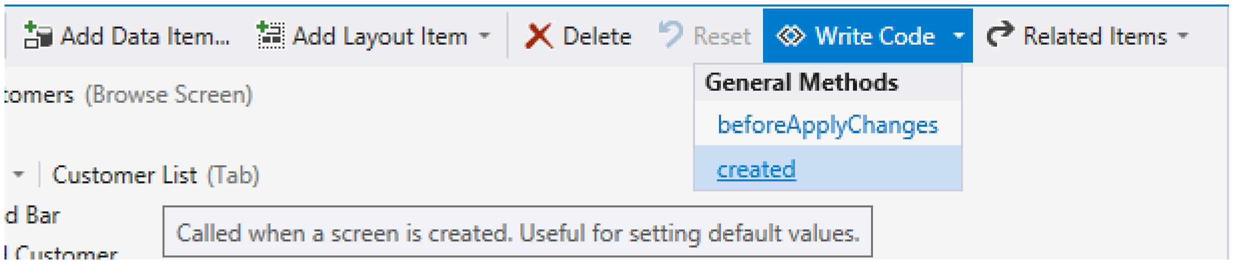
☐ Is Parameter

Format Pattern

*Figure 54: Restricting the HappinessLevel Property to a Few Choices*

The next problem to overcome is that there is no way in the screen designer to bind this integer HappinessLevel to the MinimumSatisfactionScore Query Parameter of type Percentage. You'll need to write some custom JavaScript code to accomplish this.

Find the button at the top of the screen designer called write code, and from the dropdown select the created link.



Called when a screen is created. Useful for setting default values.

*Figure 55: Creating a Function Stub*

Clicking this link will take you to the code behind file of the screen and generate a function stub that will be executed when the screen is created at run time:

```
myapp.browseCustomers.created = function (screen) {

    // Write code here.

};
```

*Code Listing 2*

Replace the body of the function with the following code snippet:

```
myapp.browseCustomers.created = function (screen) {

    screen.HappinessLevel = 0;

    screen.addChangeListener(

        "HappinessLevel",

        function () {

            screen.HapinessFactor = screen.HappinessLevel / 100;

        }

   );

}
```

*Code Listing 3*

The first line initializes the HappinessLevel screen property to 0, then adds a change listener to the screen. Whenever the HappinessLevel property changes, a callback function will be executed that sets the HappinessFactor to the percentage equivalent of the chosen HappinessLevel. Since the HappinessFactor property is still data-bound to the Query Parameter, the query will be automatically executed and the query result will be returned to the caller.

All you need to do is remove the HappinessFactor from the screen and drag the HappinessLevel property to its place. Since you restricted the HappinessLevel with a choice list, LightSwitch will automatically use a drop-down control to visualize it.

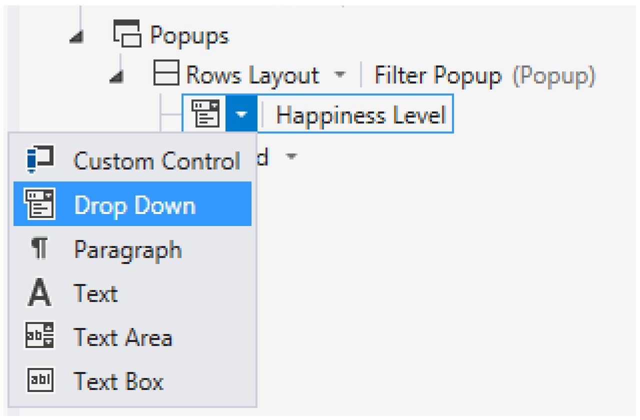
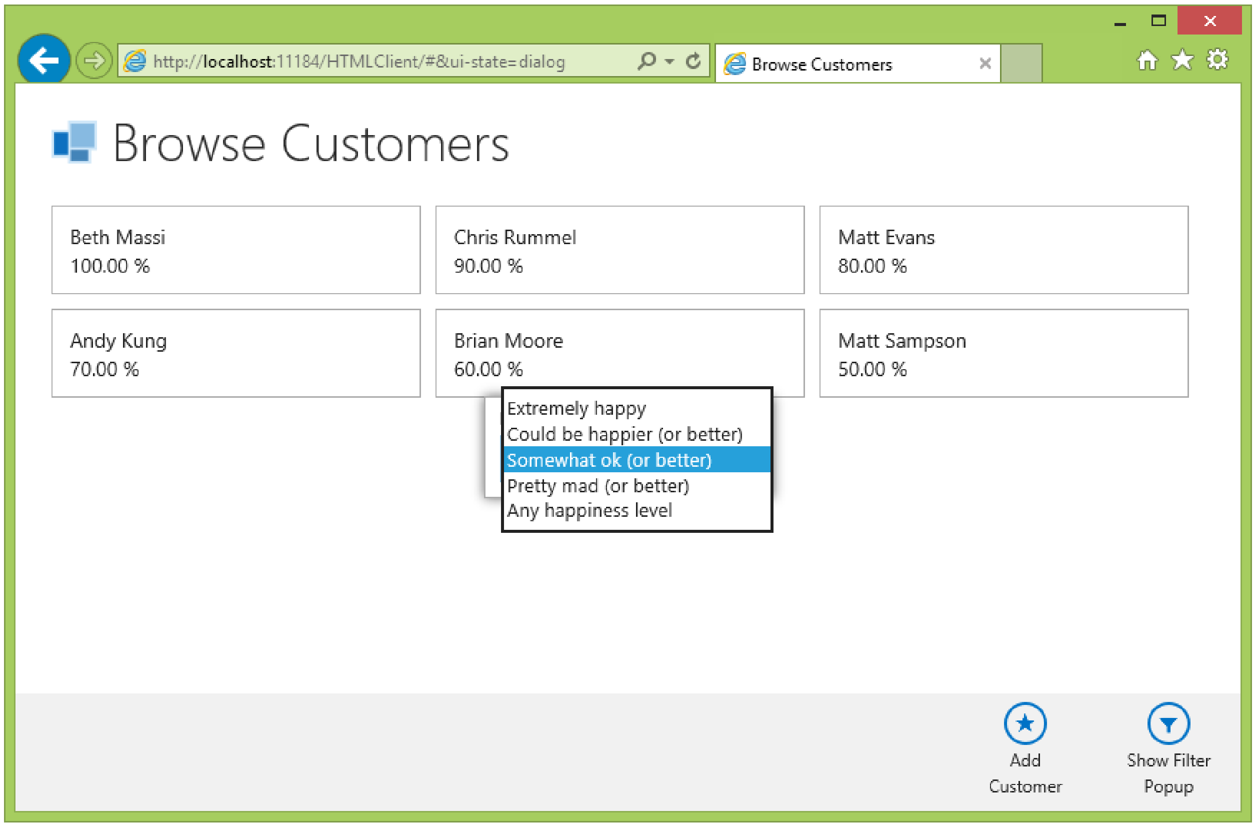


*Figure 56: Drop-down controls are preferred for restricted values.*

Save your progress and refresh the browser. The screen designer and the JavaScript do not both need to be compiled.



Beth Massi
100.00 %

Chris Rummel
90.00 %

Matt Evans
80.00 %

Andy Kung
70.00 %

Brian Moore
60.00 %

Matt Sampson
50.00 %

Extremely happy
Could be happier (or better)
Somewhat ok (or better)
Pretty mad (or better)
Any happiness level

*Figure 57: An Implemented Filter*

## Custom controls: PostRendering

There are a lot of different screen layouts and actions that require no code at all but still allow you to tweak your application to the needs of the end user.

Sometimes though, it makes sense to slightly alter the way that LightSwitch visualizes particular elements. It is possible to augment or even completely override the outcome of the LightSwitch view engine at any point. The former is a process known to LightSwitch developers as PostRendering.

A first example could be to color the background based on the gender of a customer.

Back in the screen designer, select the rows layout ContentItem in the view just below the tile list. This ContentItem is databound to a particular customer, as you can see both in the screen designer and in the properties window.

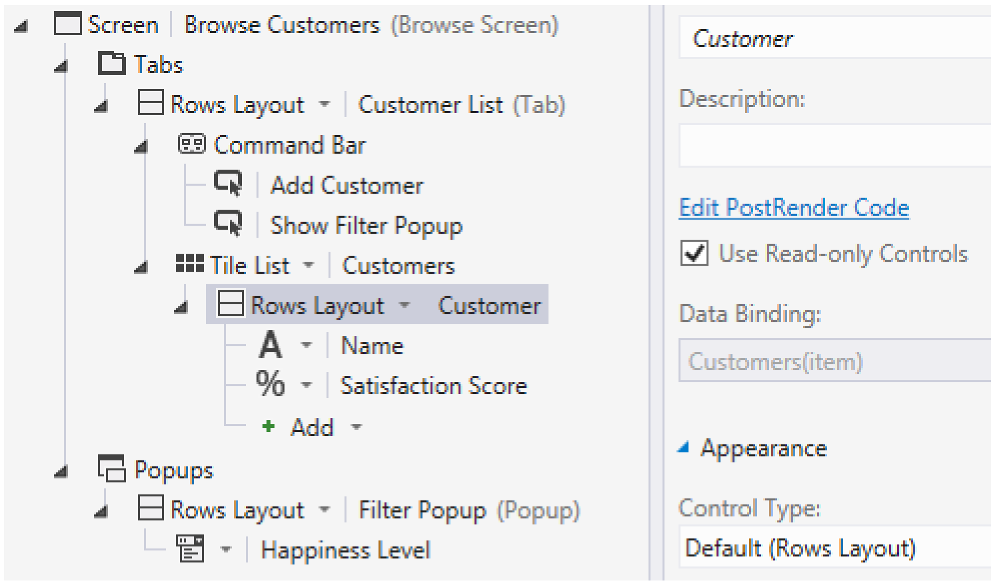


*Figure 58: The rows layout ContentItem in the visual tree is data-bound to a customer.*

With that element selected, in the properties window there will be a link that says "Edit PostRender Code". When you click the link, a JavaScript function stub will be generated in the same code-behind file where we already wrote some code before.

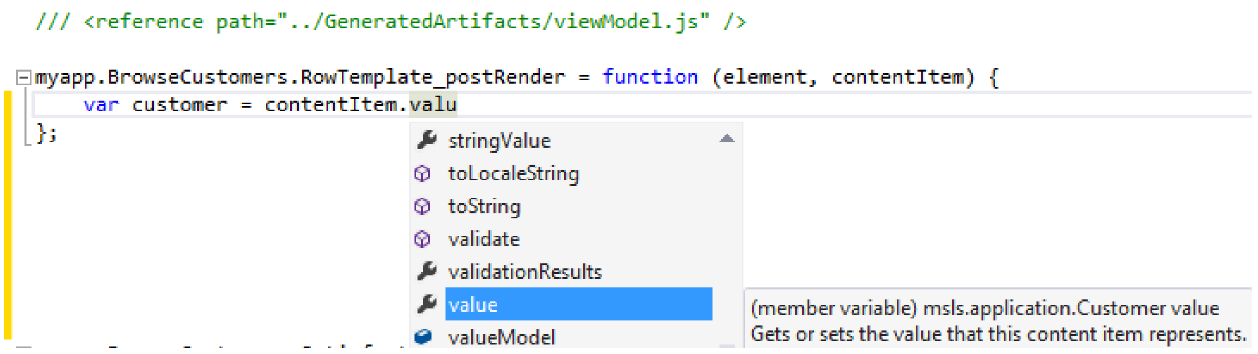
```
myapp.browseCustomers.RowTemplate_postRender = function (element, contentItem) {

    // Write code here.

};
```

*Code Listing 4*

This function receives two arguments: element and contentItem.

There really isn't a concept like *controls* in the HTML world. An HTML page is just a text file with a number of nested elements that are rendered by your browser. The argument named element is a reference to the HTML element that LightSwitch has just added to the HTML page, probably a DIV element in this case. To add, remove, or alter elements in the HTML page from JavaScript, LightSwitch will interact with a JavaScript object known as the Document Object Model (DOM).

The second argument, contentItem, is a JavaScript object that represents the Rows Layout Element node in the screen designer. You can query different properties like the DisplayName or Icon (in the case of a button), or you can get the value it is data-bound to (the customer Entity) by accessing the property called value.



```
/// <reference path="../GeneratedArtifacts/viewModel.js" />

myapp.BrowseCustomers.RowTemplate_postRender = function (element, contentItem) {
    var customer = contentItem.valu
};
```

stringValue
toLocaleString
toString
validate
validationResults
value
valueModel

(member variable) msls.application.Customer value
Gets or sets the value that this content item represents.

*Figure 59: IntelliSense on the Value Property*

Instead of accessing the customer instance to retrieve the gender through this property's value directly, it's a common practice to attach a data binding instead. This way, if the data ever changes (by the end user through the UI or even through code) the view will be automatically updated.

You already attached a data listener to the screen by using the addDataListener API. Attaching a data binding on a particular contentItem is done through a function call on contentItem called dataBind.

Just like before, this function needs two arguments: a string representing the path on the contentItem you want to bind to (value.gender in this case) and a callback function that will be executed whenever the value changes (including when the data is initially loaded).

```
myapp.browseCustomers.CustomersTemplate_postRender = function (element,
contentItem) {

    contentItem.dataBind("value.gender",

        function (gender) {

            // Code here gets executed when the Customer.gender changes...

    );

};
```

*Code Listing 5*

Let's start the coloring with a bit of static markup: setting the text color to white for readability, since you'll be coloring the background momentarily. What you need to accomplish is changing the style of the DIV element that LightSwitch inserted. Instead of interacting with the DOM directly, we'll use a JavaScript library named jQuery.

If you are completely new to jQuery, I recommend *jQuery Succinctly* from Syncfusion. However, if you are reading this e-book during your only coffee break today, here's the brief version that you'll need in order to understand the code used in the rest of this e-book:

Different browsers will sometimes have different ways in which you have to interact with the HTML document from JavaScript. jQuery is an open-source JavaScript library designed to simplify client-side scripting by adding helper methods for common tasks and by abstracting the formerly mentioned browser anomalies behind a simplified, unified API.

jQuery selectors are an example of this (http://www.w3schools.com/jquery/jquery_ref_selectors.asp). Using a jQuery selector actually means executing the JQuery() function to get a reference to a jQuery object that wraps a single element or even multiple HTML elements in a simplified adapter. To make matters even more confusing for the starting JavaScript developer, there is an alias to the JQuery() function: the dollar sign. When you see: $("div"), just know that this equals JQuery("div"), and that this will return a jQuery wrapper around all HTML DIV elements currently present in the page (the DOM).

Doing LightSwitch customizations, you'll typically use these jQuery selectors with a number of different arguments to produce various effects. Finding all elements of a particular type or with a particular ID is one way, but you can also use the jQuery selector to create new HTML elements from scratch, with code such as:

```
    var customElement = $("<div />");
```

*Code Listing 6*

You'll be creating new elements soon, but for now you'll just want to wrap jQuery around the DIV element that LightSwitch inserted so that you can access some jQuery utility functions to alter the style that is used. Surprisingly, the argument named element can also be passed directly to the JQuery function. Thus our postRender function becomes:

```
myapp.browseCustomers.CustomersTemplate_postRender = function (element,
contentItem) {

    contentItem.dataBind("value.gender",

        function (gender) {

            $(element). // Wrapped a jQuery object around the element



    );

};
```

*Code Listing 7*

One of the easiest ways to alter the style of a particular element is by directly altering the CSS through the CSS function on the jQuery object:

```
myapp.browseCustomers.CustomersTemplate_postRender = function (element,
contentItem) {

    contentItem.dataBind("value.gender",

        function (gender) {

            $(element).css("color", "white");

    );

};
```

*Code Listing 8*

The same trick can be used to alter the background based on the customer's gender as well:

```
myapp.browseCustomers.CustomersTemplate_postRender = function (element,
contentItem) {

    contentItem.dataBind("value.gender",

        function (gender) {

            $(element).CSS("color", "white");

            if (gender == "F")

                $(element).parent('li').CSS("background", "#EE317C");

            else

                $(element).parent('li').CSS("background", "#131083");

            }

    );

};
```

*Code Listing 9*

To color the background of the tile,  navigate upwards in the DOM to find a matching list item (LI) element by using the jQuery .parent function to avoid a gray border inside the tile, as described in the MSDN Leading LightSwitch column http://msdn.microsoft.com/en-us/magazine/dn160191.aspx.

Save your work and refresh the browser to see the colors in place.

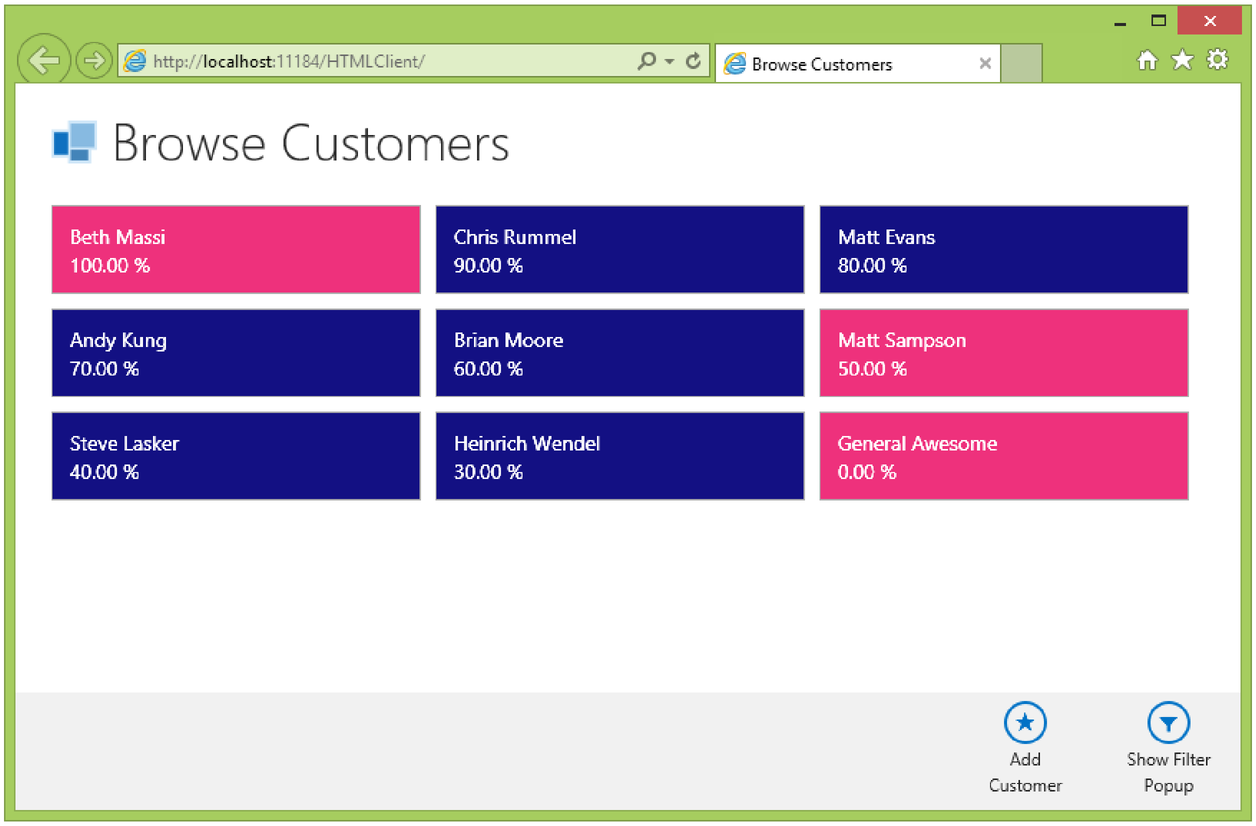


*Figure 60: Tiles Colored by Gender*

Although it is possible to alter the CSS directly from the code, programming like this doesn't result in an application that is easy to maintain. Instead, it is a better approach to apply a particular style (called a class in the HTML world) in code, then tweaking this class from separate CSS files.

## Custom controls: Rendering

Rendering is exactly the same as PostRendering, only instead of tweaking the result of LightSwitch by adding elements to the DOM, you are completely overriding and taking control of this yourself. As an example, we're going to change the application so that it displays the customer's satisfaction score not as text but as an icon.

When it comes to finding or creating good icons, I highly recommend downloading Syncfusion's free Metro Studio from http://www.syncfusion.com/downloads/metrostudio.

If you search Metro Studio for smiley, more than enough suitable icons are displayed.

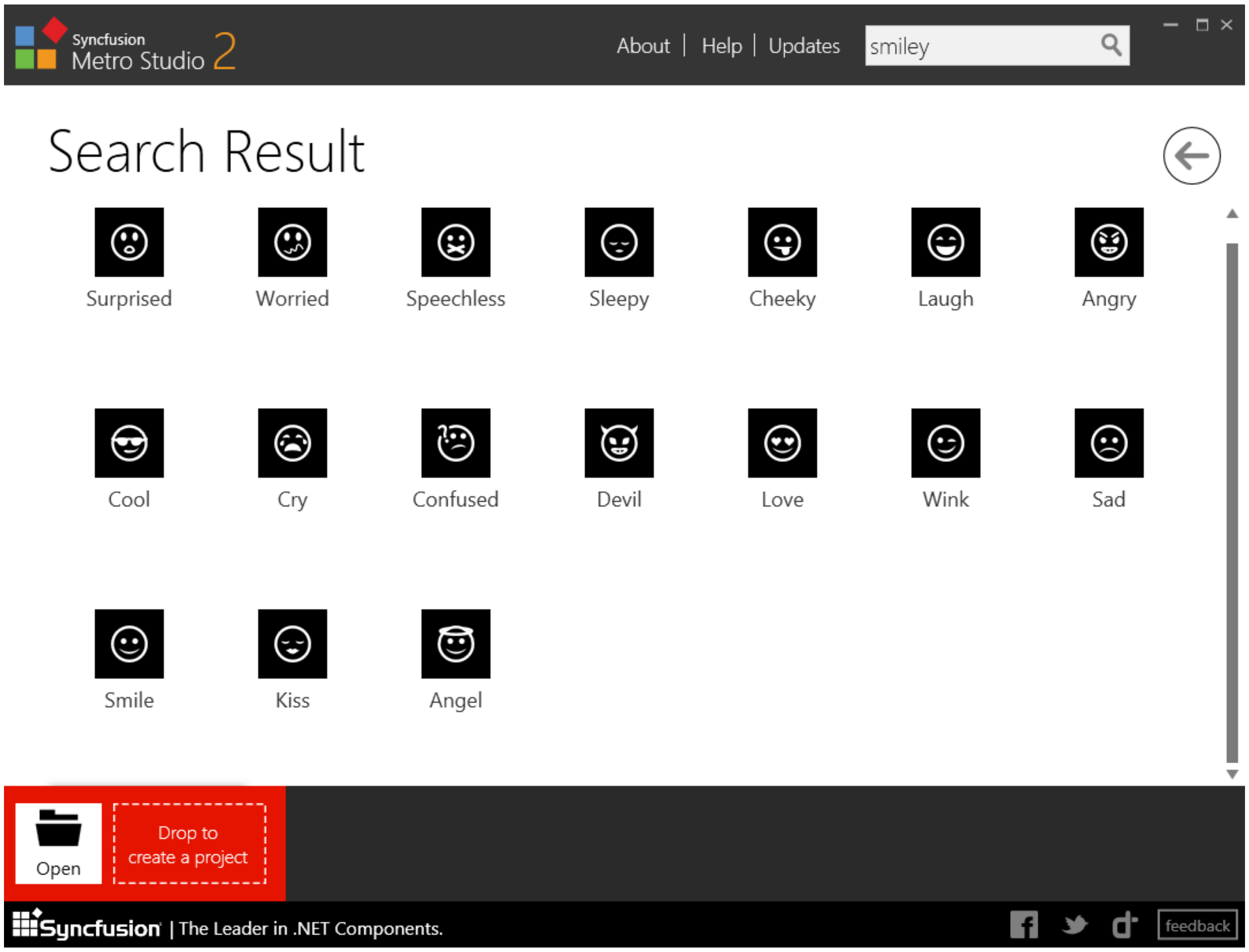
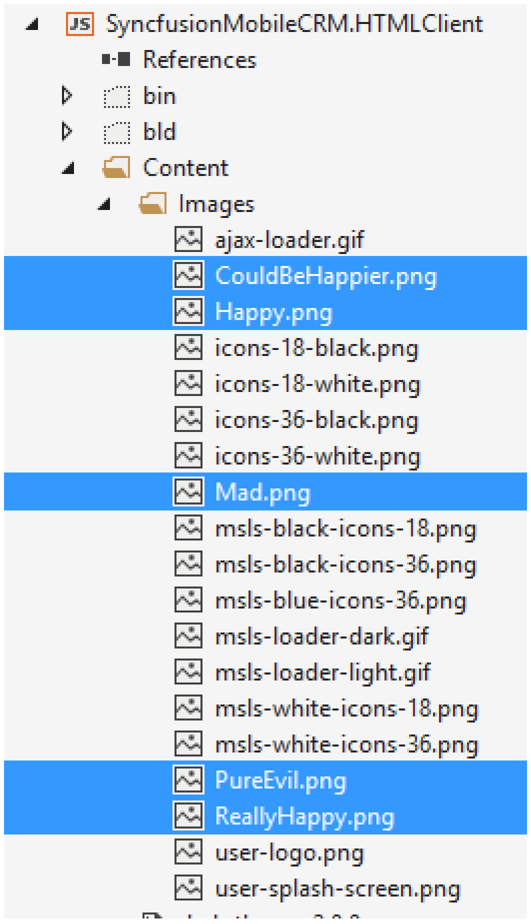


*Figure 61: Smileys in Syncfusion Metro Studio*

I took five of them and exported them to the images folder (located inside the content subfolder in the HTML project) with names varying from ReallyHappy.png to PureEvil.png.



```
◢ SyncfusionMobileCRM.HTMLClient
     References
   ▷ bin
   ▷ bld
   ◢ Content
      ◢ Images
           ajax-loader.gif
           CouldBeHappier.png
           Happy.png
           icons-18-black.png
           icons-18-white.png
           icons-36-black.png
           icons-36-white.png
           Mad.png
           msls-black-icons-18.png
           msls-black-icons-36.png
           msls-blue-icons-36.png
           msls-loader-dark.gif
           msls-loader-light.gif
           msls-white-icons-18.png
           msls-white-icons-36.png
           PureEvil.png
           ReallyHappy.png
           user-logo.png
           user-splash-screen.png
```

*Figure 62: Content and Images Subfolders in Solution Explorer*

By the way, one of the images, user-logo.png, is the icon sitting at the top left of each screen. Feel free to take this opportunity to swap it out for your company logo.

Besides images, the content subfolder is the designated place to store your cascading style sheets (CSS). One of the CSS files is named user-customization.CSS

This file allows you to override any styling that LightSwitch created without having to actually modify the LightSwitch proprietary CSS (and thus your customizations become more resilient to versioning). For smaller applications, I use this user-customization.CSS file not only to alter styling but to add new classes as well. Open the user-customization file and add the following styling:

```
ReallyHappyCustomer, .HappyCustomer, .CouldBeHappierCustomer, .MadCustomer,
.PureEvilCustomer {

    width: 48px;

    height: 48px;

}
```

```
.ReallyHappyCustomer {
    background-image: url(Images/ReallyHappy.png);
}
.HappyCustomer {
    background-image: url(Images/Happy.png);
}
 .CouldBeHappierCustomer {
    background-image: url(Images/CouldBeHappier.png);
}
 .MadCustomer {
    background-image: url(Images/Mad.png);
}
 .PureEvilCustomer {
    background-image: url(Images/PureEvil.png);
}.
```

*Code Listing 10*

The classes I added have names ranging from ReallyHappyCustomer to PureEvilCustomer. All five of them simply set the width, height, and background image. Now all that's left to do is create an element and apply these classes from code.

Open the browse customer screen in the screen designer.

Originally, each Tile consisted of a Rows Layout that vertically rendered out the customer's name and satisfaction score using a text element and a percentage viewer.

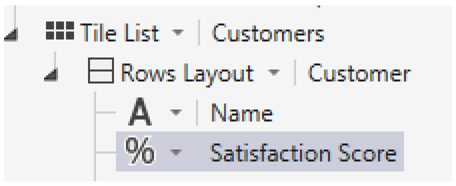


*Figure 63: Original Configuration*

Change the layout of the tiles to a columns layout to horizontally render the satisfaction score using a custom control and then the Name.

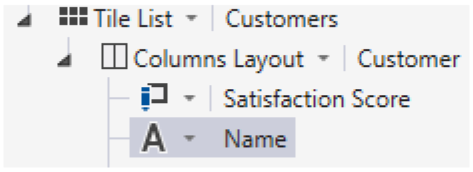


*Figure 64: Revised Configuration*

Select the name ContentItem and set the text alignment to center and width to stretch to container.

When you select the satisfaction score node in the screen designer, the properties window will not have the link to generate a PostRender function, but instead a link called edit render code:

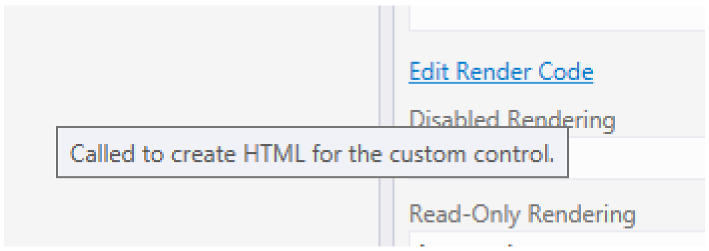


*Figure 65: Custom controls have an edit render code link in the properties window.*

Clicking that link generates a familiar function stub in the same JavaScript code-behind as the PostRender method did:

```
myapp.browseCustomers.SatisfactionScore_render = function (element,
contentItem) {

    // Write code here.

};
```

*Code Listing 11*

Again, the function has two arguments named element and contentItem. Element is a reference to an HTML DIV tag that serves as the placeholder around your custom control. ContentItem is again a JavaScript object that references the LightSwitch node from the visual tree, which is databound to the customer's SatisfactionSore.

Our JavaScript task will be to add a custom HTML element to the DOM using JQuery, then setting the appropriate CSS class (from ReallyHappyCustomer to PureEvilCustomer) based on the value of the customer's satisfaction score.

Adding a custom element will again be accomplished using the JQuery $ function, this time passing it a bit of HTML, then appending it to the parent element that was passed as an argument:

```
myapp.browseCustomers.SatisfactionScore_render = function (element,
contentItem) {

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

    customElement.appendTo(element);

};
```

*Code Listing 12*

Next, use the LightSwitch databinding API again to attach a callback function to the value of the contentItem (the satisfaction score):

```
myapp.browseCustomers.SatisfactionScore_render = function (element,
contentItem) {

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

    customElement.appendTo(element);



    contentItem.dataBind("value",

        function (satisfactionScore) {

           // You will add the appropriate CSS class here.

        }

     );
};
```

*Code Listing 13*

Finally, in that callback function, apply the appropriate CSS class using the addClass() function on the JQuery object:

```
myapp.browseCustomers.SatisfactionScore_render = function (element,
contentItem) {


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

    customElement.appendTo(element);


    contentItem.dataBind("value",

        function (satisfactionScore) {

            customElement.removeClass();

            if (satisfactionScore > 0.8)

                customElement.addClass("ReallyHappyCustomer");

            else if (satisfactionScore > 0.6)

                customElement.addClass("HappyCustomer");

            else if (satisfactionScore > 0.4)

                customElement.addClass("CouldBeHappierCustomer");

            else if (satisfactionScore > 0.2)

                customElement.addClass("MadCustomer");

            else

                customElement.addClass("PureEvilCustomer");

        }

     );
};
```

*Code Listing 14*

Did you notice the following line, which clears all the CSS classes on the element?

```
            customElement.removeClass();
```

*Code Listing 15*

I did this because the actual value of the customer's satisfaction score can change during the execution of the application. When this happens (even when done in another screen or from code), our callback function will be executed again. Clearing the CSS classes on our custom element this way helps to avoid duplicate or conflicting styling.

Save the progress and refresh the browser to test the new styling.

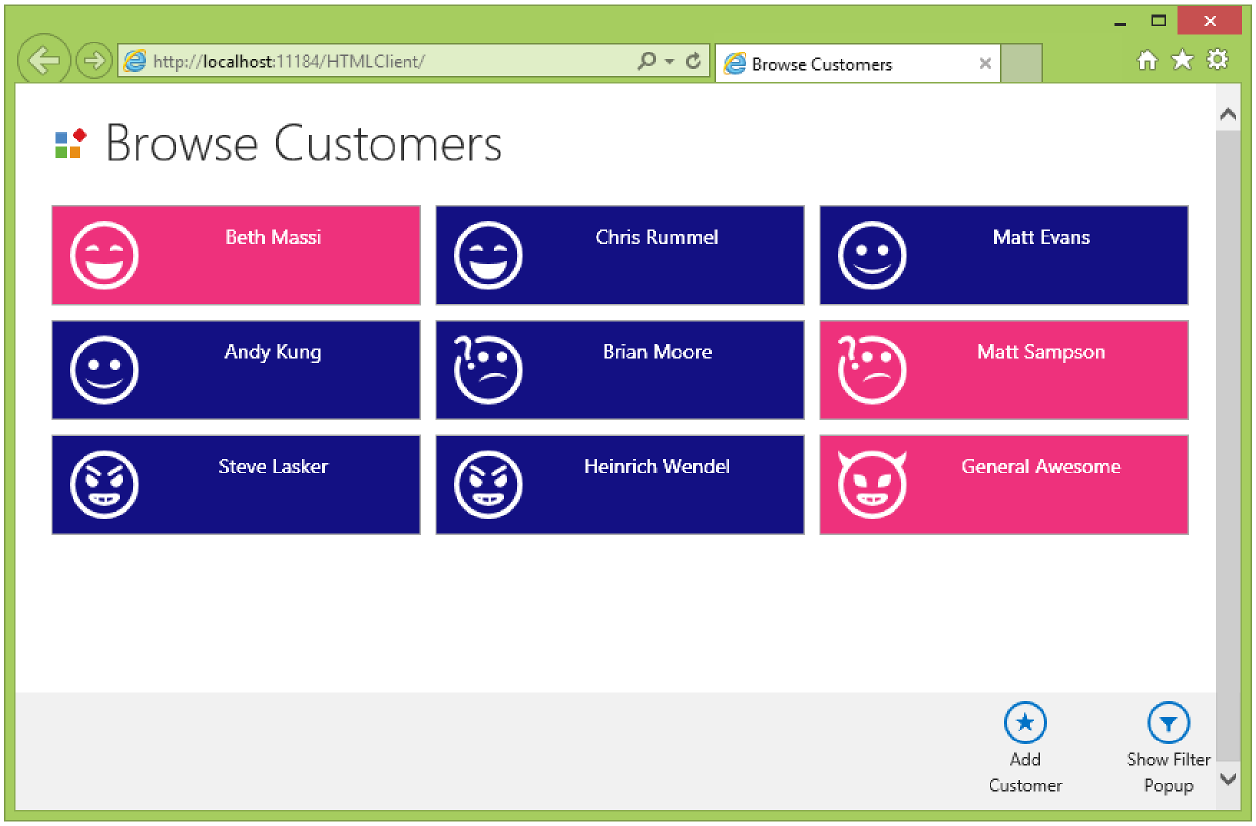


*Figure 66: Customers' satisfaction score styled as icons*

# Chapter 5  Advanced Programming Principles

## Crossing navigation boundaries.

With the browse customer screen formatted, it is time to turn our attention to the add edit Customer screen.

The first task is to transform the screen from the small popup it is now into a screen that takes up the full available height and width. Open the screen and select the root node in the visual tree, then from the properties window clear the Show As Dialog checkbox.

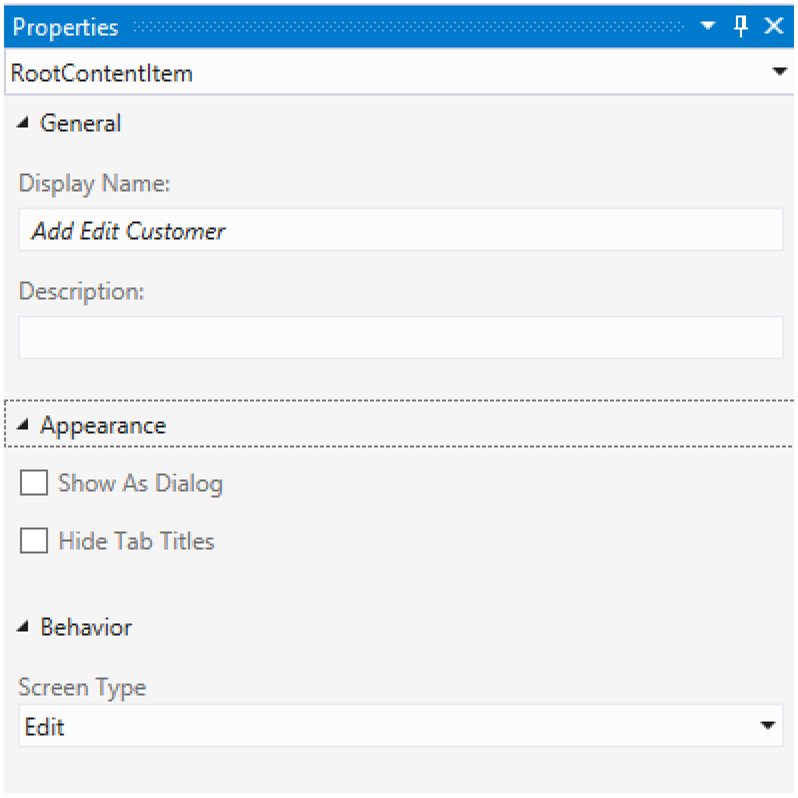


*Figure 67: Properties window with the Screen's Root Element Selected*

Besides appearance properties, you'll also find a Behavior property called Screen Type which is used to indicate whether the screen is an Edit or a browse screen.

LightSwitch uses a design pattern called *unit of work*, which means that changes are accumulated into a single transaction, and then sent to the server to be validated and processed as a whole.  Each time you navigate from an edit to another screen, the current unit of work will be passed to that screen implicitly, and any changes made in that screen will be committed or discarded together with the current changes.

However, when you navigate from one browse screen to the next, a new unit of work will be started. This implies that if there are any pending changes, LightSwitch will display a popup to ask the end user to commit or discard the current changes prior to nagivating, or to cancel navigation and Stay on Page all together.

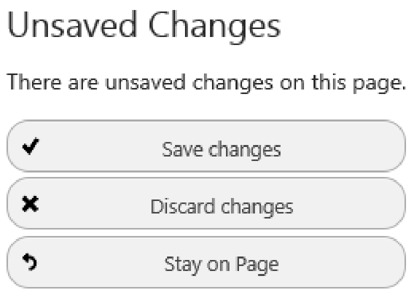


*Figure 68: Unsaved Changes Dialog*

You as a developer can also explicitly close a unit of work. From any JavaScript function, this would be done by calling one of three methods on the global myApp object:

```
myapp.applyChanges();

myapp.commitChanges();

myapp.cancelChanges();
```

*Code Listing 16*

The first function, applyChanges(), will first try to validate the pending changes and, when they are valid, launch a request to process the unit of work asynchronously on the server.

CommitChanges() does the same thing, but after a positive response from the server, it will close the current screen and navigate back to the previous screen.  CancelChanges() also navigates back asynchroneously discarding the open unit of work.

Every action that you can do from the screen designer can also be invoked from JavaScript, and more, usually by invoking functions on the global myapp object. Just as an example, the following snippet demonstrates how you would navigate to the add-edit customer screen from code:

```
myapp.showAddEditCustomer(null);
```

*Code Listing 17*

This showAddEditCustomer method was generated on the global myapp object, when you created the screen from the IDE. As an argument it expects a customer entity. To open the screen to edit a new customer entity, you might think about writing code like this:

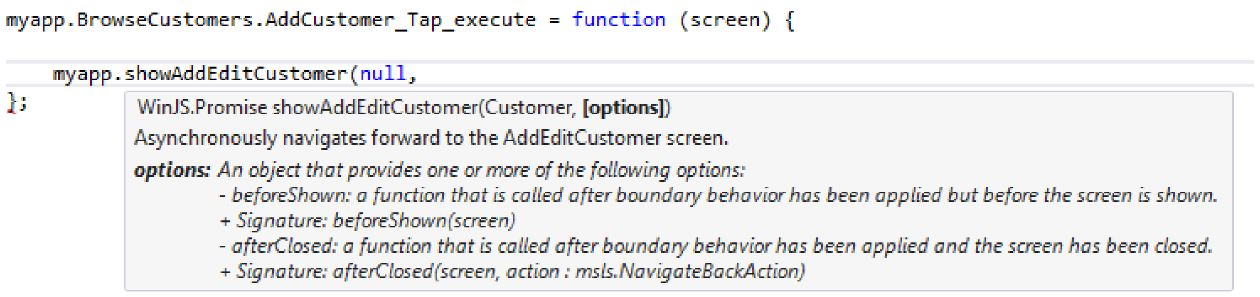
```
myapp.showAddEditCustomer( new msls.application.Customer());
```

*Code Listing 18*

However, if you would actually try to run this code from a button on the browse customer screen, you would run into some interesting behavior. As soon as you click the button that executes this code, LightSwitch prompts that you have pending changes that need to be saved or discarded before navigating to the add-edit customer screen.

What actually happens is that the constructor of the customer entity is called within the boundaries of the unit of work belonging to the browse customer screen. Because you're moving from one screen to another, the unit of work needs to be completed or discarded, resulting in the prompt to save or cancel.

Luckily, the showAddEditCustomer method accepts a second argument that allows you to add one callback function that will be executed before the screen is shown, and one that is executed after the screen has been closed:



```
myapp.BrowseCustomers.AddCustomer_Tap_execute = function (screen) {

    myapp.showAddEditCustomer(null,
};
```

WinJS.Promise showAddEditCustomer(Customer, **[options]**)
Asynchronously navigates forward to the AddEditCustomer screen.
***options:*** *An object that provides one or more of the following options:*
*- beforeShown: a function that is called after boundary behavior has been applied but before the screen is shown.*
*+ Signature: beforeShown(screen)*
*- afterClosed: a function that is called after boundary behavior has been applied and the screen has been closed.*
*+ Signature: afterClosed(screen, action : msls.NavigateBackAction)*

*Figure 69: Passing Callback Functions when Navigating to a Screen*

Reading the IntelliSense on the JavaScript immediately reveals the missing piece of the puzzle: the beforeShown callback function is executed within the boundaries of the add-edit customer screen. To open the screen to edit a new customer entity you would need to write:

```
myapp.showAddEditCustomer(null,

{

    beforeShown: function (detailScreen) {

        detailScreen.Customer = new msls.application.Customer();

    }

});
```

*Code Listing 19*

The beforeShown callback function is mostly used to perform scenario-specific initialization code; the afterClosed callback function is useful to perform any action after the detail screen has finished its unit of work and has been closed, perhaps even depending on how the user left the screen (was the unit of work committed or cancelled), for example:

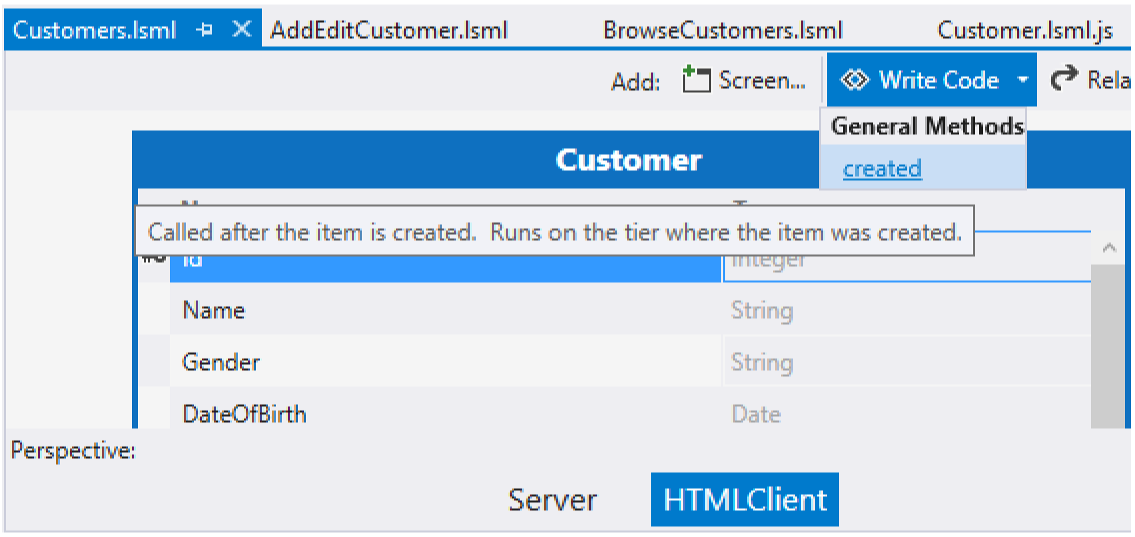
```
myapp.showAddEditCustomer(null,

{

    beforeShown: function (detailScreen) {

        detailScreen.Customer = new msls.application.Customer();

        detailScreen.Customer.SatisfactionScore = 0.85;

    },

    afterClosed: function (detailScreen, navigationResult) {

        if (navigationResult == msls.NavigateBackAction.cancel)

            msls.showMessageBox("User cancelled out of changes.");

                                                                            else

                                                screen.showPopup("NewUserPopup");

                                                                               }

});
```

*Code Listing 20*

## Initializing values

Initializing values is usually done in three distinct spots: in the created event that fires when an entity is created, using the beforeShown callback function when navigating to a screen from code, or in the created event that fires when a screen is created, in this order.

To initialize values on a particular entity, you need to open that entity in the entity designer, select the perspective of the HTMLClient at the bottom, then click Write Code at the top, and select the created event to generate a function stub.



Called after the item is created. Runs on the tier where the item was created.

*Figure 70: The Write Code Button for Entities*

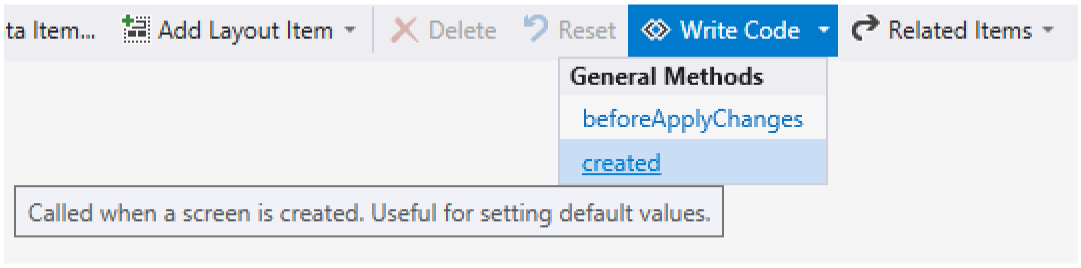
Suppose that on average, a customer starts out as a happy customer, resulting in a hypothetical SatisfactionScore of 80%. You could model this in the appliction in this created() function stub that was generated from the entity designer:

```
myapp.Customer.created = function (entity) {

    entity.SatisfactionScore = 0.8;

};
```

*Code Listing 21*

However, per screen you could still initialize this value to something different if desired.

From the write code button in the screen designer, each screen has a similar option to generate a function stub that gets executed when the screen is first created.



Called when a screen is created. Useful for setting default values.

*Figure 71: The Write Code Button for Screens*

This function can be used to do general initialization of any property, value, or entity on the screen. You could use this function stub to test if the customer that's being edited is a new or existing entity, and then set the screen's title from the default add-edit customer to something more appropriate:

```
myapp.AddEditCustomer.created = function (screen) {

    if (screen.Customer.Id) {

        screen.details.displayName = screen.Customer.Name;

    }

    else {

        screen.details.displayName = "Add a new customer.";

    }

};
```

*Code Listing 22*

At this point, a business rule could dictate that any new customers created via this screen start with a basic satisfaction score of 90%:

```
myapp.AddEditCustomer.created = function (screen) {

    if (screen.Customer.Id) {

        screen.details.displayName = screen.Customer.Name;

    }

    else {

        screen.details.displayName = "Add a new customer.";

        screen.Customer.SatisfactionScore = 0.9;

    }

};
```

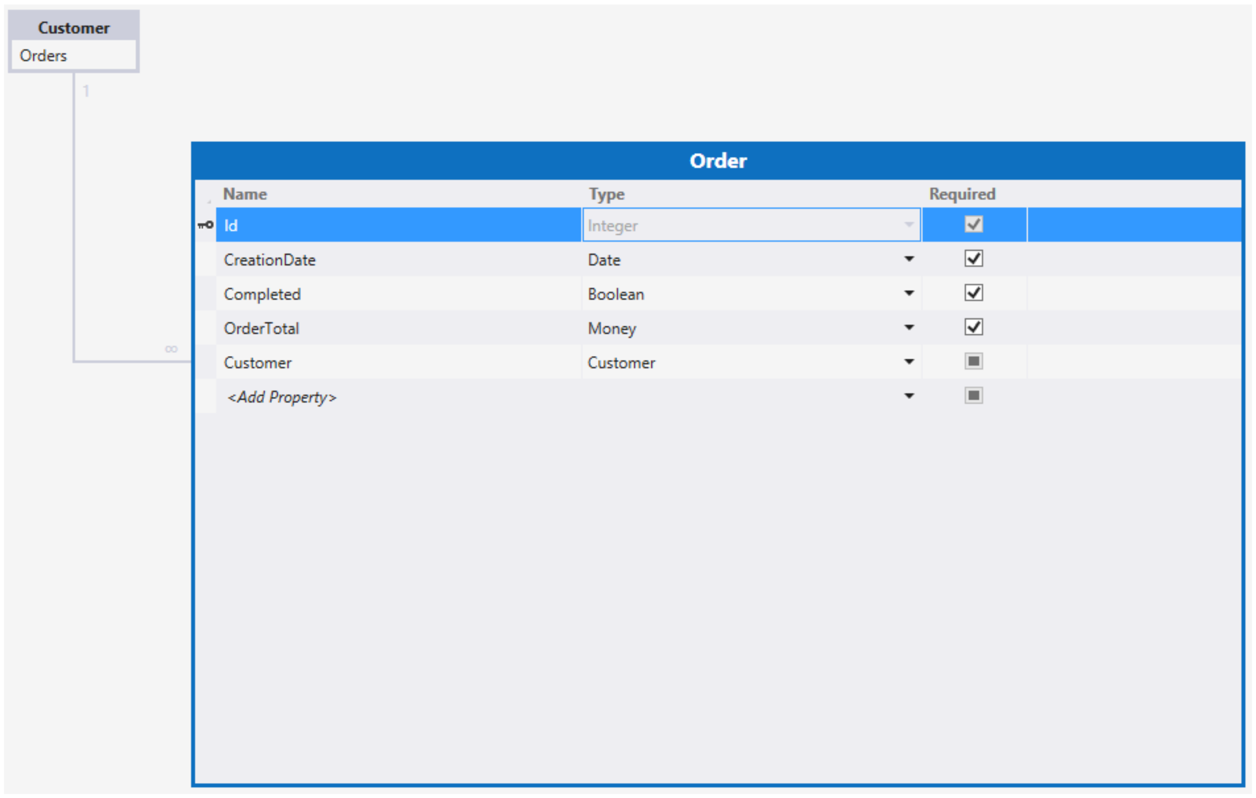
*Code Listing 23*

Remember that this code snippet runs after the beforeShown callback if the screen was opened from code and you specified such a method.

## Asynchronous programming

The initialization code that we wrote for the add-edit customer screen was pretty straightforward: set the DisplayName of the screen and initialize the satisfaction score of the customer entity. We didn’t touch any properties or fields that could potentially take a while to complete. When you do, to prevent the interface from freezing up while lengthy code is executing, LightSwitch will force you to write asynchronous JavaScript code using WinJS ‘promise’ objects. As an exercise, we’re going to try to show a message on the add-edit customer screen that displays whether or not the customer has any open orders.

This of course implies we’re going to extend the application by introducing a new entity called Order, with a one-to-many relationship between order and customer.



Order

| Name | Type | Required |
|---|---|---|
| Id | Integer | ✓ |
| CreationDate | Date | ✓ |
| Completed | Boolean | ✓ |
| OrderTotal | Money | ✓ |
| Customer | Customer | ■ |
| <Add Property> | | ■ |

*Figure 72: The Order Entity*

I also added hundreds of these Order entities using the same SQL Post Deployment Script that was added in an earlier chapter.

You can download the full script from [https://gist.github.com/janvanderhaegen/6682208](https://gist.github.com/janvanderhaegen/6682208) to get the sample data yourself.

```sql
SET IDENTITY_INSERT [dbo].[orders] ON;
```

```
MERGE INTO [dbo].[orders] AS Target

USING (VALUES

      (1, '20121103', 1, 238, 2),

-- more random orders ...

      (298, '20120903', 1, 105, 2),

      (299, '20110220', 1, 1689, 3)

)

AS Source(Id, CreationDate, Completed, OrderTotal, Order_Customer)

ON Target.Id = Source.Id

-- update matched rows

WHEN MATCHED THEN

UPDATE SET CreationDate = Source.CreationDate, Completed = Source.Completed,
OrderTotal = Source.OrderTotal,

      Order_Customer = Source.Order_Customer

-- insert new rows

WHEN NOT MATCHED BY TARGET THEN

INSERT (Id, CreationDate, Completed, OrderTotal, Order_Customer)

VALUES (Id, CreationDate, Completed, OrderTotal, Order_Customer)

-- delete rows that are in the target but not the source

WHEN NOT MATCHED BY SOURCE THEN

DELETE;


SET IDENTITY_INSERT [dbo].[orders] OFF;


GO
```

*Code Listing 24*

If all went well, you can now open the add-edit customer screen in the screen designer again and spot the new link on the customer entity on the view model (left-hand side) to add the customer's orders to the screen view model.

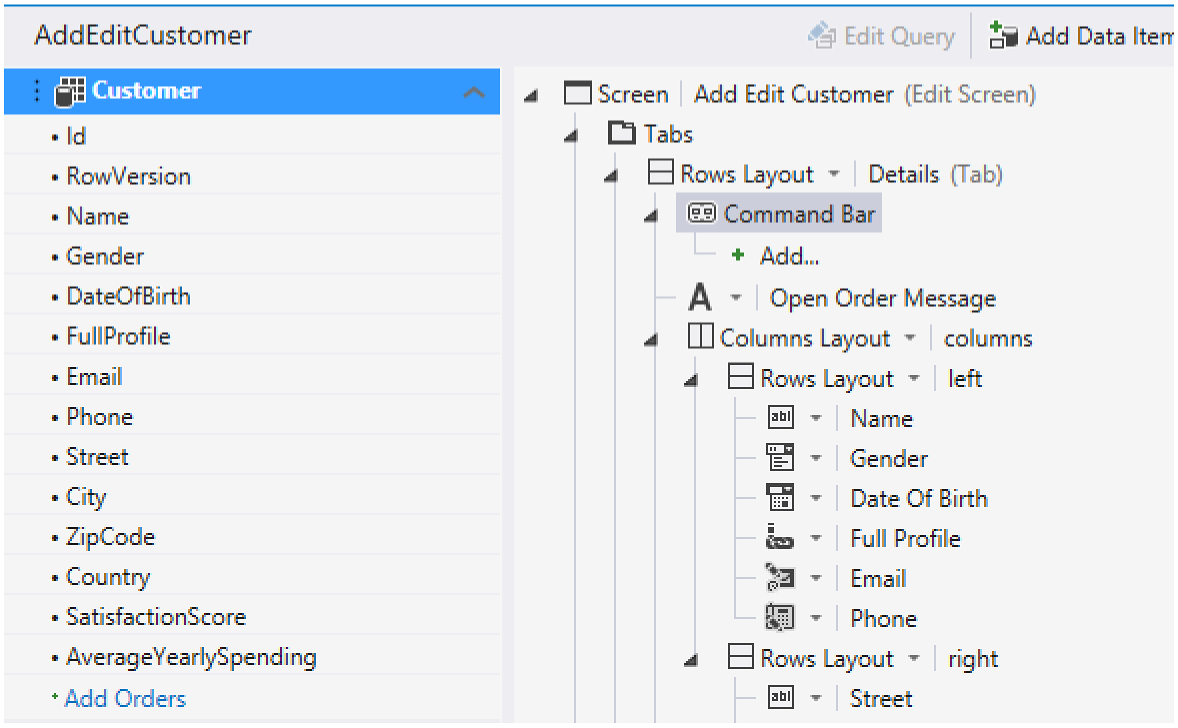


*Figure 73: You can add the customer's orders to the screen with Add Orders.*

Once you click Add Orders, LightSwitch will add a new VisualCollection to the screen view model named Orders.

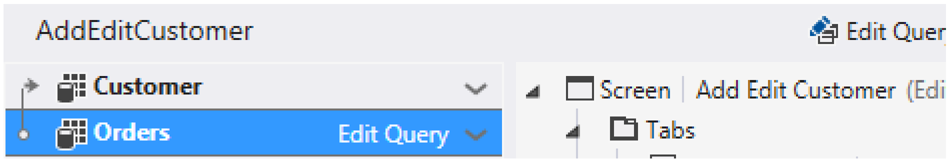


*Figure 74: The Order VisualCollection and the Customer Entity*

Click the edit query link to open the orders in the query designer and adjust it to only retrieve the orders that havent't been completed yet.

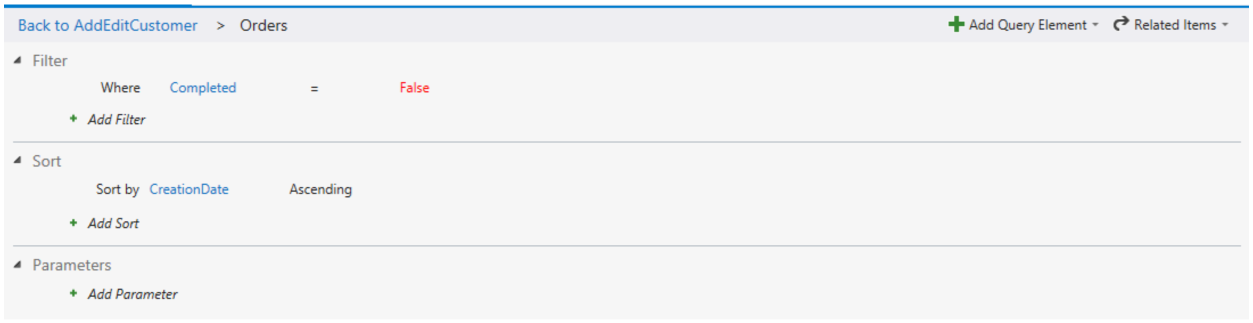


*Figure 75: Retrieving Open Orders Only Via the Query Designer*

Rename the orders collection to OpenOrders.

Now that the application has the notion of orders and you have added sample data and a collection of open orders to the screen, you've got everything in place to display a message if the customer currently has any pending orders.

To be able to show this message, we'll need to store it somewhere on the view model. Add a new string property to the view model by clicking the add data item button at the top of the screen designer, and creating the OpenOrderMessage property in the add data item wizard.

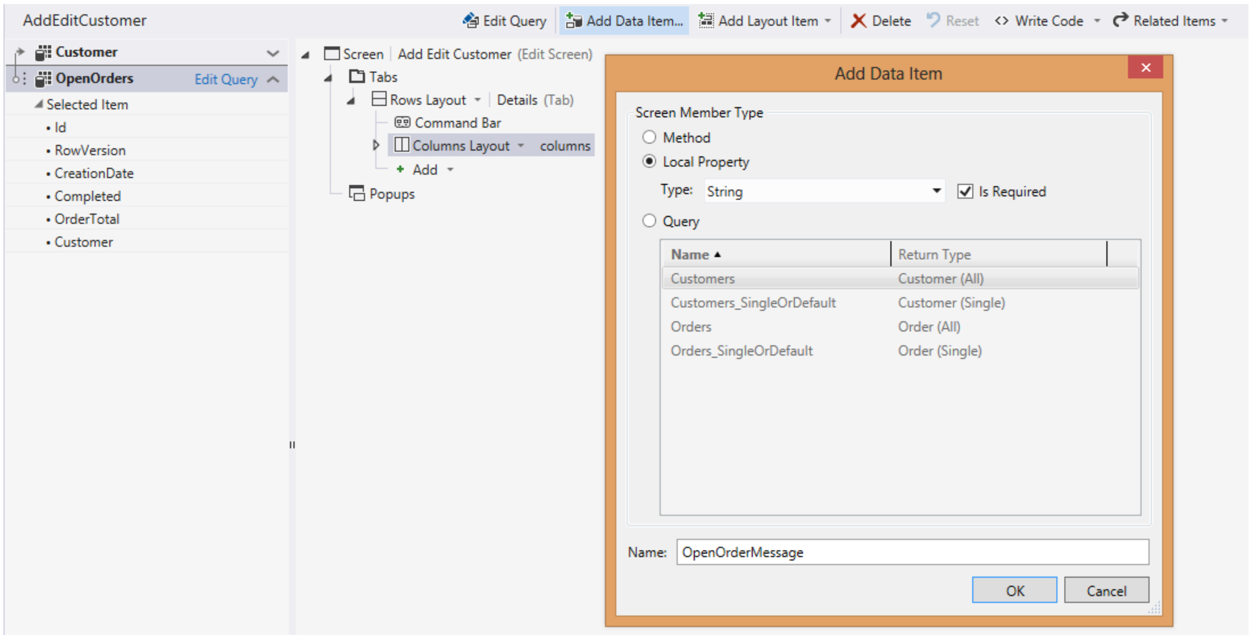


*Figure 76: The add data item wizard allows you to add a local property.*

From the view model on the left-hand side, drag this new string property somewhere on the view.

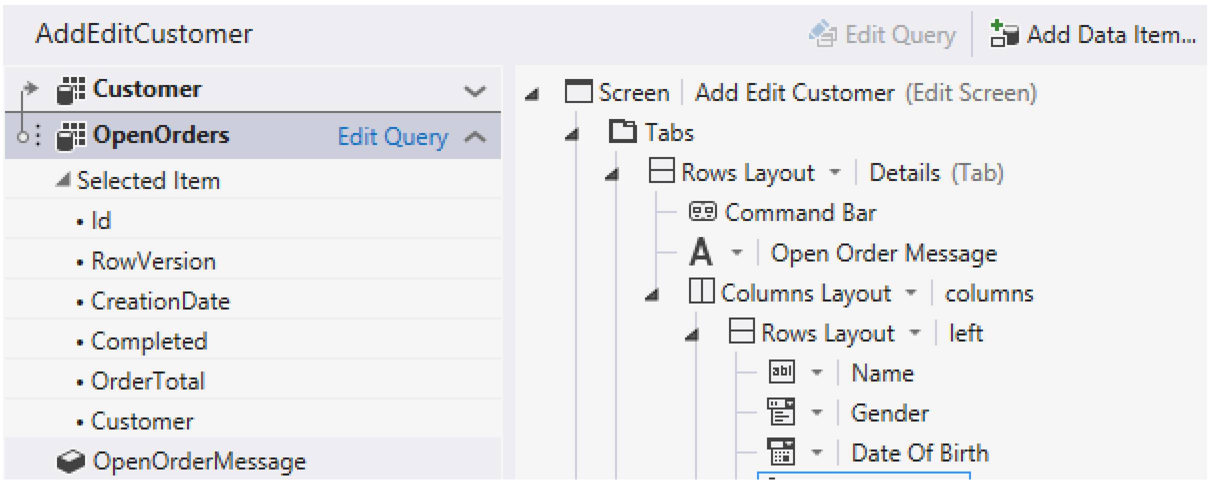


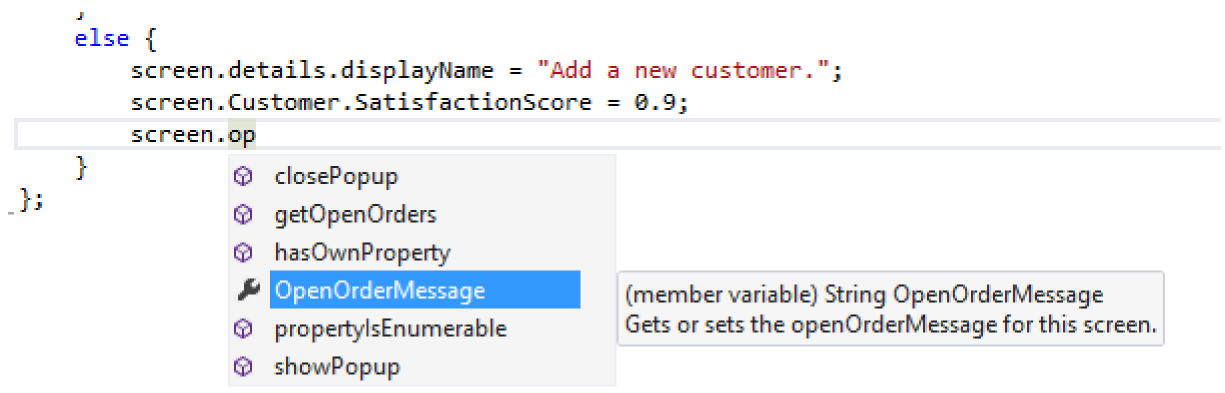
*Figure 77: The OpenOrderMessage Property on the View Model and in the View*

Lastly, we need to store a message in that OpenOrderMessage property. You can do that from the screen's created function, which currently looks like:

```
myapp.AddEditCustomer.created = function (screen) {
    if (screen.Customer.Id) {
        screen.details.displayName = screen.Customer.Name;
    }
    else {
        screen.details.displayName = "Add a new customer.";
        screen.Customer.SatisfactionScore = 0.9;
    }
};
```

*Code Listing 25*

The screen's view model has a customer and OpenOrders and OpenOrderMessage properties. However, the JavaScript IntelliSense only reveals the customer entity and OpenOrderMessage string to use as a property from within JavaScript.

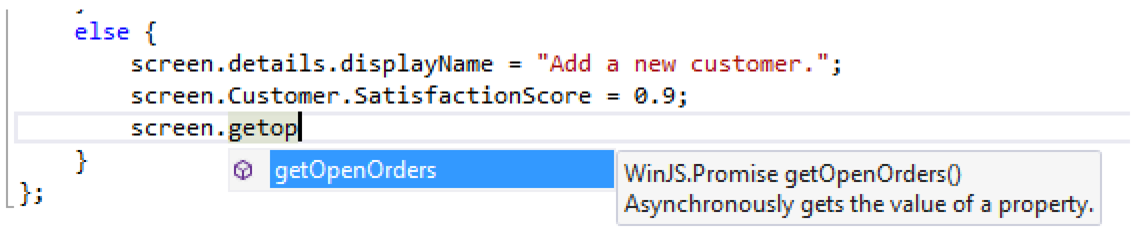


```
    else {
        screen.details.displayName = "Add a new customer.";
        screen.Customer.SatisfactionScore = 0.9;
        screen.op
    }
};
```

(member variable) String OpenOrderMessage
Gets or sets the openOrderMessage for this screen.

*Figure 78: A JavaScript property was generated for OpenOrderMessage but not for OpenOrders.*

Instead of an OpenOrders property, a function was generated named getOpenOrders, which returns a WinJS promise object.



```
    else {
        screen.details.displayName = "Add a new customer.";
        screen.Customer.SatisfactionScore = 0.9;
        screen.getop
    }
};
```

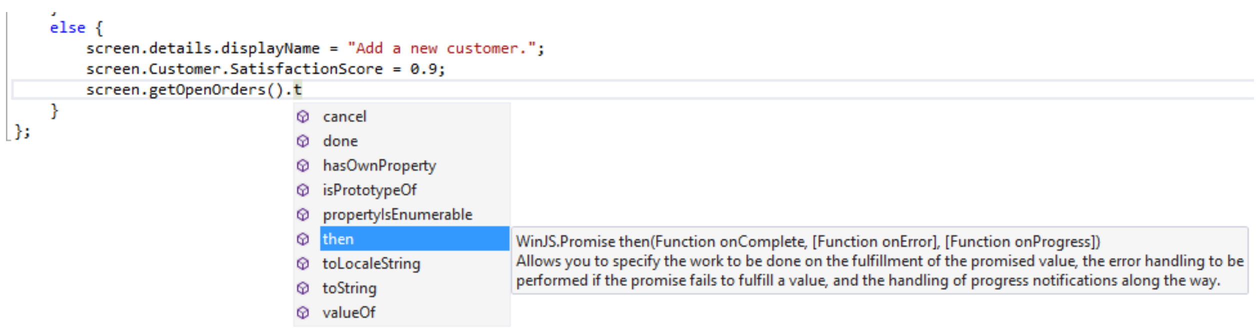
WinJS.Promise getOpenOrders()
Asynchronously gets the value of a property.

*Figure 79: The getOpenOrders Function*

The idea behind this differentiation is that LightSwitch knows the difference between a simple property that is quick to acces, and can thus be done synchronously, and a more complex property like this collection that might not be up to date or loaded at all.

The service call required to make sure the collection is up to date might take a while. Hence, LightSwitch forces you to retrieve a promise object instead of working with the collection directly.

The promise object is nothing more than a reference to the fetching of the data. IntelliSense reveals that it can be chained with a number functions, out of which the then() function is the one you'll use the most.



```
    else {
        screen.details.displayName = "Add a new customer.";
        screen.Customer.SatisfactionScore = 0.9;
        screen.getOpenOrders().t
    }
};
```

WinJS.Promise then(Function onComplete, [Function onError], [Function onProgress])
Allows you to specify the work to be done on the fulfillment of the promised value, the error handling to be performed if the promise fails to fulfill a value, and the handling of progress notifications along the way.

*Figure 80: Available Callback Functions on the Promise Object*

This then() function allows you to chain a callback function that is executed when the collection has finished loading or immediately if the collection was already loaded and up to date.

```
        screen.getOpenOrders().then(

                function (orders) {  /* Callback code goes here */}
);
```

Code Listing 26

Either way, once the callback function is executed, LightSwitch makes sure that the collection is up to date and will pass it as an argument to your callback function. The full initialization for the add-edit screen thus becomes:

```
myapp.AddEditCustomer.created = function (screen) {

    if (screen.Customer.Id) {

        screen.details.displayName = screen.Customer.Name;

        screen.getOpenOrders().then(

                function (orders) {

                    if (orders.count == 0)

                        screen.OpenOrderMessage = "No open orders";

                    else

                        screen.OpenOrderMessage = "orders pending";

                }

            );

    }

    else {

        screen.details.displayName = "Add a new customer.";

        screen.Customer.SatisfactionScore = 0.9;

        screen.OpenOrderMessage = "";

    }

};
```

Code Listing 27

Or, because a million pixels say more than a thousand words, at run time this becomes:

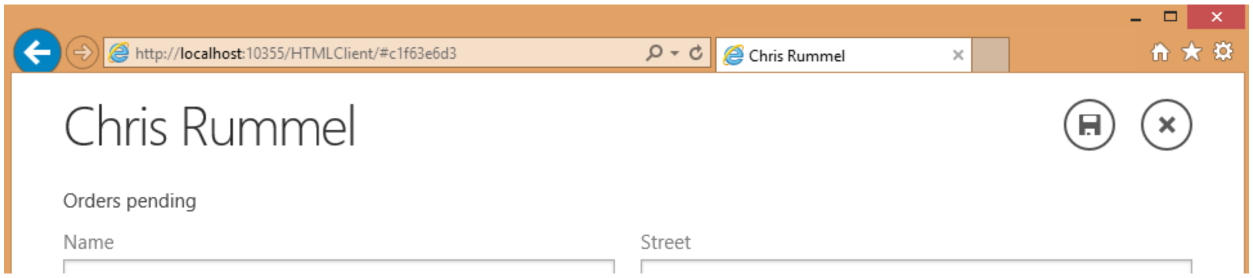


*Figure 81: The Properly Initialized Detail Screen*

***Note: It is incorrect to believe that this is a multi-threaded application. Even if these techniques are explicitlly referred to as asynchronous, JavaScript does offer multi-threading support (as per the specs: [http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf](http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf)). Instead of truly running multiple code paths at the same time (like real multi-threaded applications would), the promise objects simply encapsulate logic that alters the one single-threaded sequential code path.***

If you have a custom JavaScript function that might potentially take a while to complete, LightSwitch has a simple API call that allows you to wrap any function in a promise yourself, like Michael Washington demonstrates in the following self-describing code snippet taken from “Using Promises in Visual Studio LightSwitch” ([http://lightswitchhelpwebsite.com/Blog/tabid/61/EntryId/170/Using-Promises-In-Visual-Studio-LightSwitch.aspx](http://lightswitchhelpwebsite.com/Blog/tabid/61/EntryId/170/Using-Promises-In-Visual-Studio-LightSwitch.aspx)):

```
Myapp.PromiseOrders.created = function (entity)

    // Set the default date for the Order

    entity.OrderDate = new Date();

    // Using a Promise object you can call the CallGetUserName function

    msls.promiseOperation(CallGetUserName).then(function (PromiseResult)

        // Set the result of the CallGetUserName function to the

        // UserName of the entity

        entity.UserName = PromiseResult;

    });

};

// This function will be wrapped in a Promise object
```

```
function CallGetUserName(operation) {

    $.ajax({

        type: 'post',

        data: {},

        url: '../web/GetUserName.ashx',

        success: operation.code(function (AjaxResult)

            operation.complete(AjaxResult);

        })

    });

}
```

*Code Listing 28*

## Server-side programming

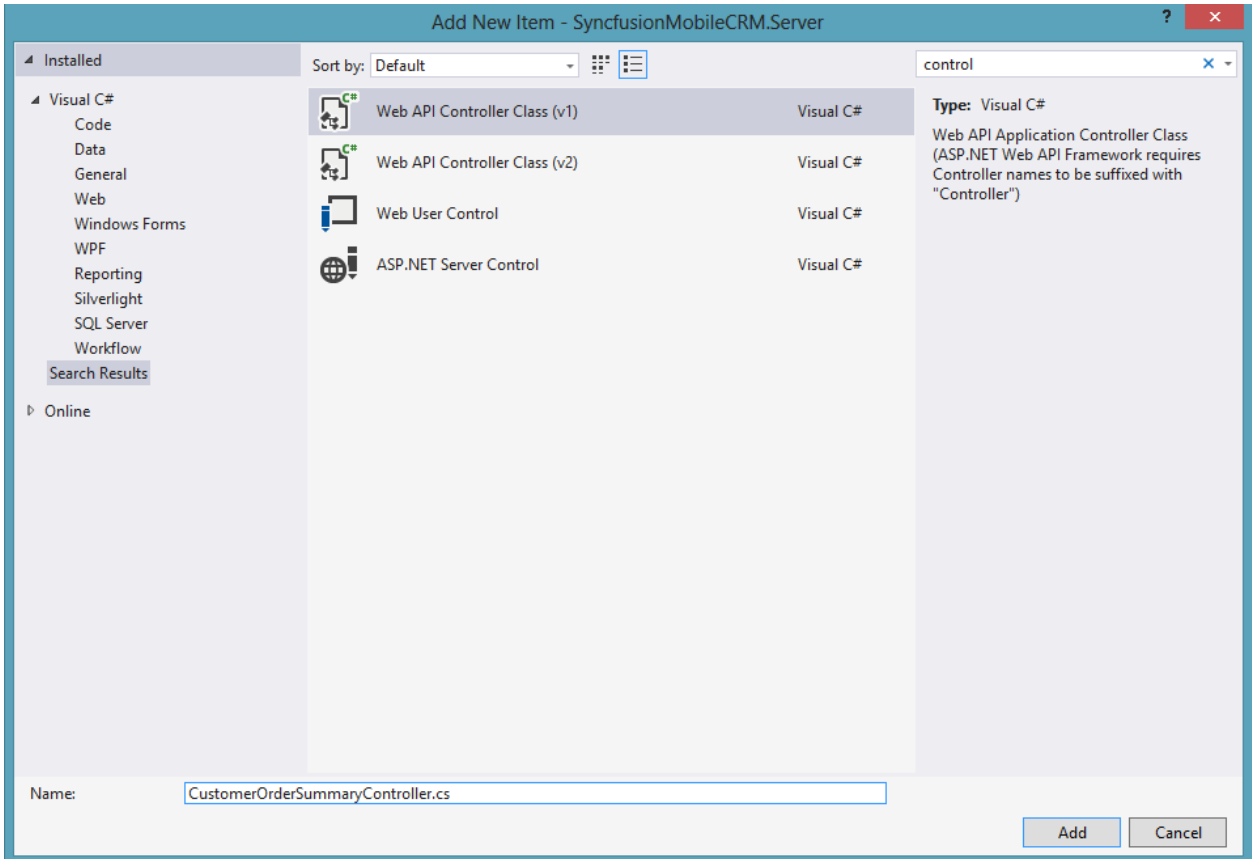
If there is at least one record, just to display it we fetched the entire collection (top 45 actually, due to paging). What a waste of the user's time. We'll need a more efficient way to program.

For these specific data queries, or to model more specific business operations, LightSwitch generates an extensive model in the server project. This way, you can accomplish your advanced scenarios in an efficient manner by adding your own WCF web services, SignalR hubs, or ASP.NET Web API.

As an illustration, let's use the last one to summarize the order history of any given customer per year.

If you are unfamiliar with ASP.NET Web API, I would suggest http://www.asp.net/web-api/overview/getting-started-with-aspnet-web-api/tutorial-your-first-web-api. Here's a brief summary. ASP.Net Web API is a server-side framework that, for any web request, the server matches the URI and HTTP verb (POST, GET, etc.) with a routing table to determine which controller should be instantiated and which method on that controller should be called to handle the request.

The required references and setup to add ASP.NET Web API are handled automatically by adding a controller, so start by right-clicking on the server project and selecting Add New Folder from the context menu. Name the folder Controllers. Add a controller by right-clicking on the folder and selecting Web API Controller from the Add New Item dialog.



| | |
|---|---|
| Web API Controller Class (v1) | Visual C# |
| Web API Controller Class (v2) | Visual C# |
| Web User Control | Visual C# |
| ASP.NET Server Control | Visual C# |

**Type:** Visual C#

Web API Application Controller Class (ASP.NET Web API Framework requires Controller names to be suffixed with "Controller")

Name: CustomerOrderSummaryController.cs

*Figure 82: Adding a Web API Controller*

The empty controller that was added will look like:

```
using System.Linq;

using System.Web.Http;


namespace LightSwitchApplication.Controllers

{

    public class CustomerOrderSummaryController : ApiController

    {

        // GET api/<controller>

        public IEnumerable<string> Get()

        {
            return new string[] { "value1", "value2" };
        }

        // GET api/<controller>/5
        public string Get(int id)
        {
            return "value";
        }
```

*Code Listing 29*

Before implementing the code, you can configure the routing for this controller so that you can give it a quick test. Right-click the server project and select Add New Item from the context menu.

Add a new global application class from the Add New Item dialog, and replace the contents with the following code:

```
using System;
using System.Web.Http;
using System.Web.Routing;

namespace LightSwitchApplication
{
    public class Global : System.Web.HttpApplication
    {
        protected void Application_Start(object sender, EventArgs e)
        {
            RouteTable.Routes.MapHttpRoute(
                name: "ReportsApi",

                routeTemplate: "reports/{controller}/{id}"

                );

        }

    }

}
```

*Code Listing 30*

The code will add a new route called the ReportsAPI, which will handle all HTTP GET requests to the URL by initializing the correct controller, and executing a method called Get() which accepts a single argument called id.

```
http://{HostURL}/reports/{NameOfTheController}/{Id}
```

*Code Listing 31*

Since you can perform  HTTP GET from your browser, you can test your setup by setting a break point in the Get method of the CustomerOrderSummaryController and pressing F5 to build and start debugging. Navigate your browser from the HTML client's default URL (http://localhost:10355/HTMLClient/) to http://localhost:10355/reports/CustomerOrderSummary/5. You should hit your break point and notice the value of the id argument is 5. The port number, 10355, is likely to be different on your system.

The next step is to implement the controller to interact with the application's server-side LightSwitch model. This is done by adding a reference to the Microsoft.LightSwitch namespace, in which LightSwitch generated a class called the ServerApplicationContext to the using statements at the top of the class. ServerApplicationContext is a new class in LightSwitch starting from VS 2012 Update 3.

Instances of the ServerApplicationContext class implement IDisposable, so they should be wrapped in a using statement for proper disposal of resources, like the connection to the database.

```
using System.Linq;

using System.Web.Http;

using Microsoft.LightSwitch;
```

```
namespace LightSwitchApplication.Controllers

{

    public class CustomerOrderSummaryController : ApiController

    {

        public object Get(int id)

        {

            using (ServerApplicationContext context =
ServerApplicationContext.CreateContext()) {



            }

        }

    }

}
```

Code Listing 32

This context exposes methods that allow access to the currently authenticated user and its permissions and access to the entities created in the entity designer. You can interact with the entities directly and even make use of some extension methods that aren't available on the client, like the GroupBy method.

> ***Remember that this GroupBy method is always available on the server, but the success of this method depends on the type of data source. You can model your LightSwitch entities around an existing SharePoint list or OData web service, in which case the grouping will fail since the underlying protocol by which LightSwitch interacts with the data source does not support grouping. Also, if the data source is a new or existing SQL or SQL Azure database, the grouping will actually be deferred to be done by the SQL server itself.***

The full code to fetch and summarize the order history based on the given ID of the customer becomes:

```
using System.Linq;

using System.Web.Http;
using Microsoft.LightSwitch;


namespace LightSwitchApplication.Controllers

{

    public class CustomerOrderSummaryController : ApiController

    {

        public object Get(int id)

        {

            using (ServerApplicationContext context =
ServerApplicationContext.CreateContext()) {

                var query = context.DataWorkspace.ApplicationData.orders

                    .Where(o => o.Customer.Id == id)

                    .GroupBy(o => o.CreationDate.Year)

                    .Select(g => new

                    {

                        Label = g.Key,

                        Value = g.Sum(o => o.OrderTotal)

                    })

                    .OrderBy(g => g.Label);


                return query.Execute().Select(

                    g => new {

                        Label = string.Format("'{0}" ,  (g.Label - 2000)),

                        Value = g.Value

                    }

                    ).ToArray();
                }
            }
        }
    }
```

Code Listing 33

From the GET operation, you are returning an array of implicitly typed CRL objects from the code. In fact, ASP.NET Web API will automatically marshall the data to JavaScript Object Notation (JSON) format, which is programming language agnostic.

You could call this from any programming language.  In fact, to test the result, feel free to press Ctrl+F5 to build and start the application (without debugging), then navigate to a hardcoded URL like http://localhost:{YourPortNumber}/reports/CustomerOrderSummary/5 to see the result. Depending on your browser, this JSON response might be shown directly or downloaded as a file.

Do you want to open or save **1.json** (160 bytes) from **localhost**?

*Figure 83: Internet Explorer offers to download the summarized order history.*

## jQuery UI Widgets: Essential Studio for JavaScript

Now that you have an efficient way to accumulate data on the server, you need an efficient way to visualize the results. Although you can write a custom control yourself using the render method, it is interesting to note that in no way does LightSwitch prevent the use of third-party libraries. This is a good opportunity to put Syncfusions's new Essential Studio for JavaScript to the test.

Essential JS is a new JavaScript framework designer for use in line of business (LOB) applications. Adding third-party components to your LightSwitch HTML application is always a three-step process. The first step is to add the required code (.js files) and styling (.css files) to the appropriate folders in the HTMLClient project. You need to make sure they are loaded by referencing them from the web page, and then you can use them where required.

After downloading the Essential JS suite, copy the stylesheets named bootstrap.CSS, default.CSS and default-responsive.CSS from the installtion directory to the Content subfolder of the HTMLClient project, and ej.widgets.all.min.js and properties.js to the Scripts subfolder. When adding a third-party script to your application, always search for a minified version, which you can recognize by the .min in the file-name, to reduce the download size.

Next up, you'll need to load these CSS and JavaScript resources into the HTML page called default.htm.

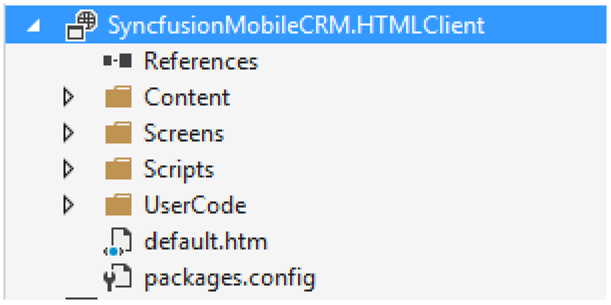


*Figure 84: Default.htm, the Single HTML Page in this SPA*

This HTML page is an empty shell, consisting of links to the required stylesheets, an HTML DIV element showing a loading animation, links to the required JavaScript files, and finally a single JavaScript block:

```
<script type="text/javascript">

    $(document).ready(function () {

        msls._run()

        .then(null, function failure(error) {

            alert(error);

        });

    });

</script>
```

*Code Listing 34*

This block will cause the Microsoft LightSwitch (msls) Javascript libraries to load and your application to start, replacing the HTML DIV element with the loading animation with the elements that form your app.

In this web page, find the links to the stylesheets and add links to the Essential JS stylesheets that you just added to the project at the bottom:

```
<!-- Syncfusion stylesheets-->

<link href="Content/bootstrap.CSS" rel="stylesheet">

<link href="Content/default.CSS" rel="stylesheet" />
```

```
    <link href="Content/default-responsive.CSS" rel="stylesheet" />
```

*Code Listing 35*

Next, find the links to the JavaScript files and add links to the Essential JS JavaScript files:

```
    <!-- Syncfusion widgets-->

    <script src="Scripts/ej.widgets.all.min.js"
type="text/javascript"></script>

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

*Code Listing 36*

When adding third-party JavaScript, it's a good practice to add references to the end, but just above the generatedAssets import:

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

*Code Listing 37*

Finally, to avoid conflicting duplication, you have to comment out the first script reference:

```
    <!--<script type="text/javascript"
src="//ajax.aspnetcdn.com/ajax/4.0/1/MicrosoftAjax.js"></script>-->
```

*Code Listing 38*

Now that the required Essential JS files have been added to the project and loaded into the HTML page, you can use them as custom controls in the application.

First, open the add-edit customer screen in the screen designer and create a new tab called OrderSummary. On this tab, drag the customer's id property from the view model onto the view and use a custom control to render it. From the properties window, set the label position to none.

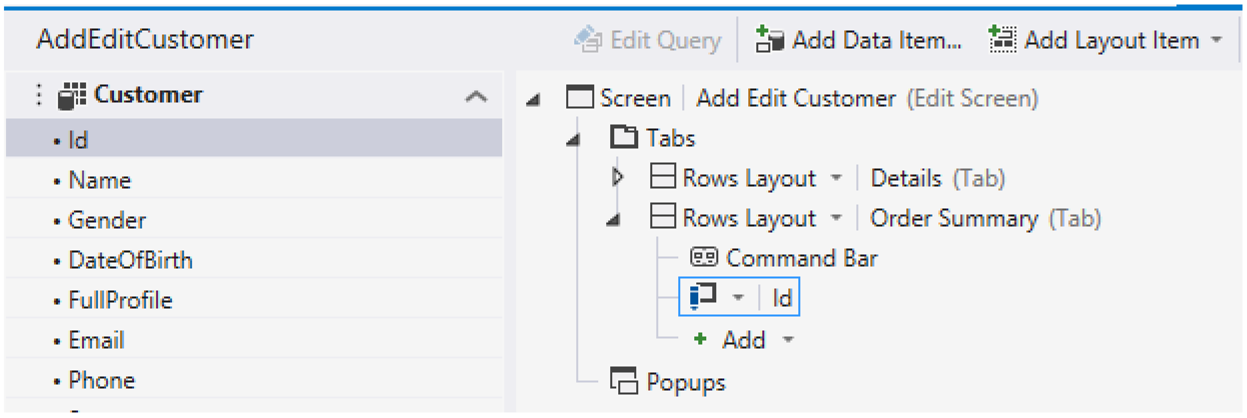


*Figure 85: Adding a New Tab to the Screen*

From the properties window, click the edit render code link to generate a function stub like you did earlier. Again, you'll start by writing some code that inserts a DIV element into the HTML page, then data-bind to the value (the customer ID) with a callback function:

```
myapp.AddEditCustomer.Customer_Id_render = function (element, contentItem) {

    $(element).append('<div id="container" style="width:700px" />');

    contentItem.dataBind(

                                                                     "value",

                                                        function (customerId) {



                                                                             }

                                                                            );

};
```

*Code Listing 39*

In that callback function, you'll use a jQuery selector to wrap a jQuery object around the DIV element you just inserted by passing a hash tag and the ID:

```
           $("#container")
```

*Code Listing 40*

Almost any JavaScript library will be wrapped in a jQuery UI widget.  This means that the producer of the JavaScript library registered the HTML controls with jQuery's plugin system by using a specialized jQuery function. The advantage is that in this way the producer can add functionality to the jQuery Object directly. For example, this is how to turn the DIV with the container ID into an Essential JS chart:

```
            $("#container").ejChart()
```

*Code Listing 41*

Obviously you'll need to pass additional parameters to this initialization method The most important one will be to set the data source. You'll instruct this Essential JS chart to fetch its data directly from the ASP.NET Web API controller that you created in the previous chapter:

```
            $("#container").ejChart(

                {

                    series: [{

                        dataSource: {

                            data: new ej.DataManager({

                                url: "../reports/CustomerOrderSummary/" +
customerId,



                            }),


```

*Code Listing 42*

The full code includes additional formatting and coloring:

```
myapp.AddEditCustomer.Customer_Id_render = function (element, contentItem) {

    $(element).append('<div id="container" style="width:700px" />');

    contentItem.dataBind(

        "value",

        function (customerId) {

            $("#container").ejChart(
                {
                    chartAreaBorder: {
                        width: 1
                    },
                    primaryXAxis:
                    {
                        rangePadding: 'Additional',
                        title: {  },
                    },

                    primaryYAxis:
                    {
                        title: { text: "Total (in USD)" },
                    },

                    series: [{
                        name: ' ', type: 'column',
                        animation: true,
                        dataSource: {
                            data: new ej.DataManager({
                                url: "../reports/CustomerOrderSummary/" +
customerId,

                            }),
                            xName: "Label",
                            yNames: ["Value"],
                            query: ej.Query()
                        },

                        style: { interior: "#7ED600" }
                    }],
                    load: "loadTheme",
                    size: { height: 470 },
                    legend: { visible: false },
                }
            );
        }
    );
};
```

*Code Listing 43*

Save your progress and refresh your browser to see the result in action: the add-edit screen for any customer now has two tabs. Click the header to open the order summary tab, which will display a graphic of the data served by the ASP.NET Web API.

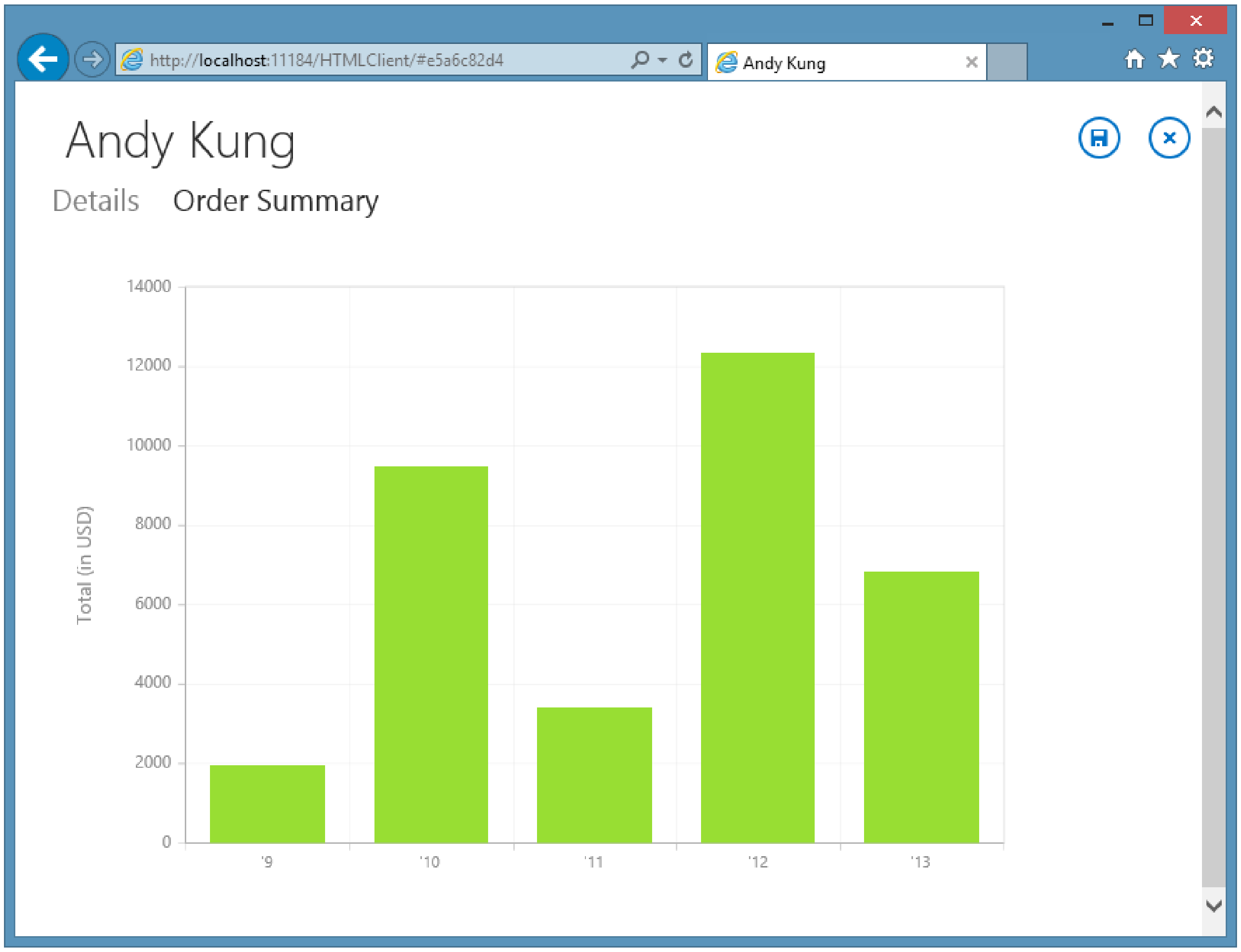


*Figure 86: Essential JS in Action in the LightSwitch Application*

When running the application, you will see a beautiful animation in action.

## jQuery UI Widgets: custom Bing Maps control

It would be a shame to end this chapter without demonstrating how you can take advantage of some device-specific capabilities like geolocation. Geolocation is the name used to represent a JavaScript built-in function to request information about the current position, altitude, or velocity. It's a JavaScript API, so it is compatible with every mobile device. The JavaScript engine will actually ask the OS, which will first ask permission from the user.

To test this, we're going to add functionality that help users navigate from their current location to the address listed for the current customer.

Start by adding a new tab to the add-edit customer screen called address tab, then add a new command to the command bar.

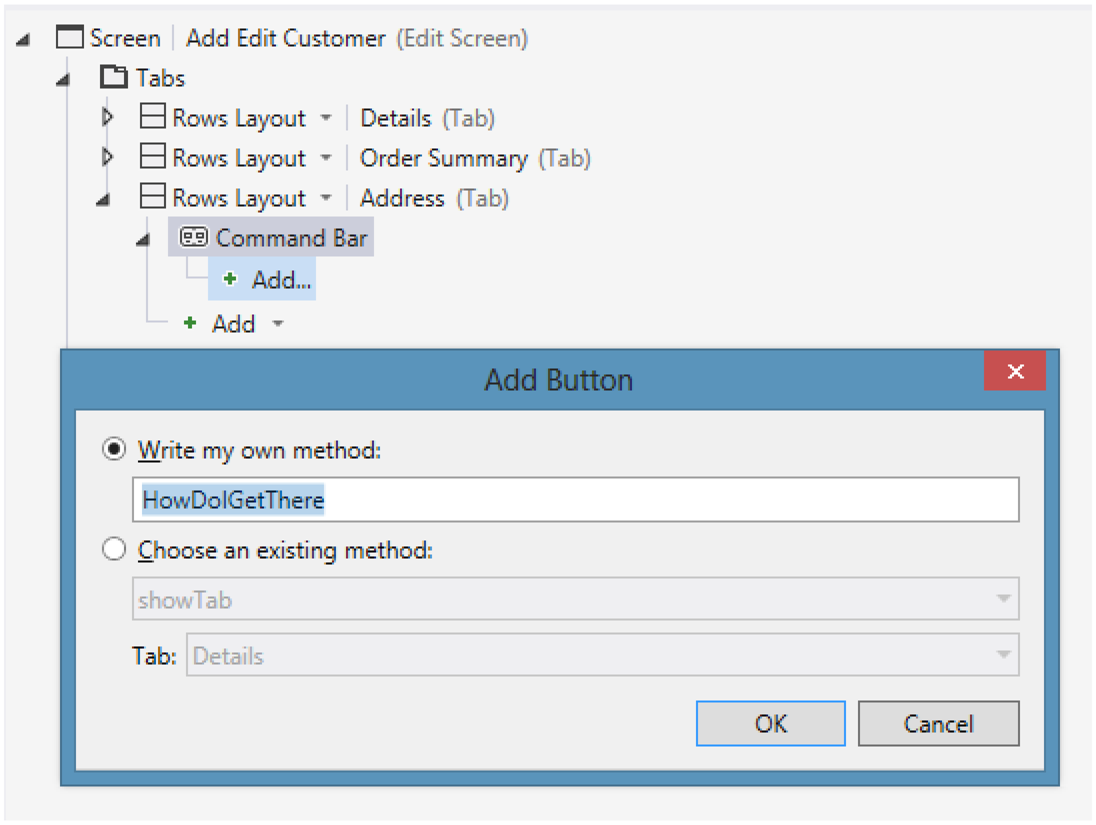

Screen | Add Edit Customer (Edit Screen)
Tabs
Rows Layout | Details (Tab)
Rows Layout | Order Summary (Tab)
Rows Layout | Address (Tab)
Command Bar
Add...
Add
Add Button
Write my own method:
HowDoIGetThere
Choose an existing method:
showTab
Tab: Details
OK
Cancel

*Figure 87: Generating Custom Methods from Add Button Wizard*

Generate a custom method called HowDoIGetThere. This will generate the button in the tab's command bar in the view and as a corresponding command in the ViewModel.

Select the newly added command from the properties window and change the icon to Question. Then, right-click on the command and select the Edit Execute code option.

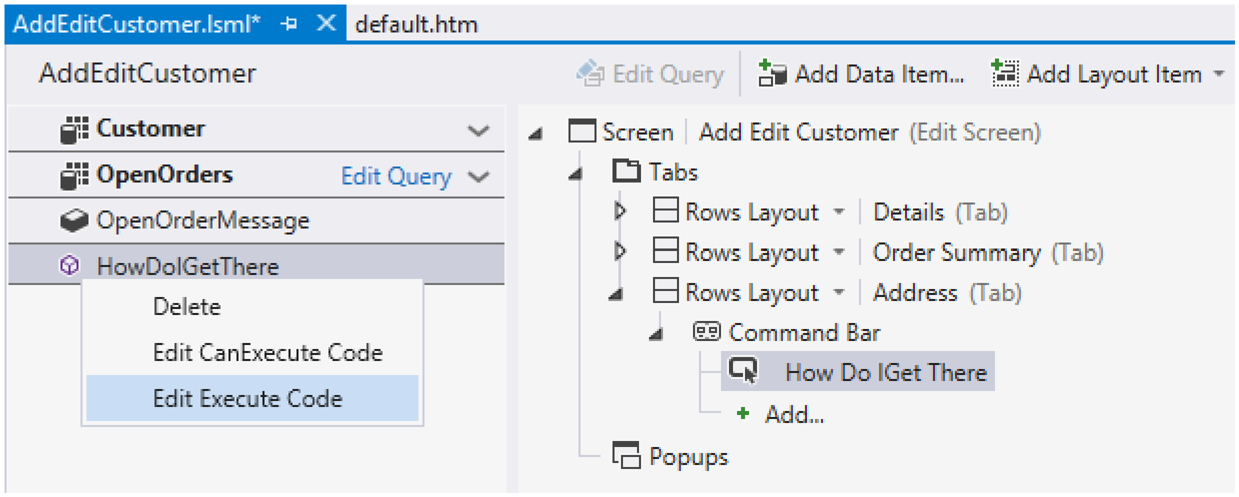


*Figure 88: Context Menu for Custom Commands*

Replace the generated method stub with the following code:

```
myapp.AddEditCustomer.HowDoIGetThere_execute = function (screen) {

    if (navigator.geolocation) {

        navigator.geolocation.getCurrentPosition(function (position) {

            msls.showMessageBox("Current location is " +
position.coords.latitude + "," + position.coords.longitude);

        });

    }

    else { msls.showMessageBox("Geolocation is not supported by this
browser."); }

};
```

*Code Listing 44*

The code will check if geolocation is supported, and then ask for the current location—which will require permission from the end user—and display it in a message box.

Instead of displaying the current coordinates in a message, we're going to show a map with detailed instructions on how to get to the customer using another jQuery UI widget: https://gist.github.com/janvanderhaegen/6310722.

***The widget itself is based on a post by LightSwitch program manager Heinrich Wendel: http://blogs.msdn.com/b/lightswitch/archive/2013/01/14/visualizing-list-data-using-a-map-control-heinrich-Wendel.aspx.***

After downloading the widget, make sure you add it to the project and reference it in the default.htm page. Open the add-edit customer screen again and from the left-hand side of the screen designer drag the customer onto the view.

Select Custom Control as the control to use and set the label position to none again.

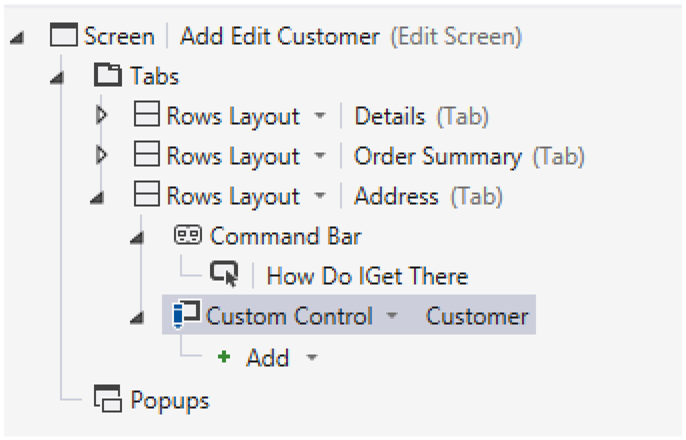

Screen | Add Edit Customer (Edit Screen)
Tabs
Rows Layout | Details (Tab)
Rows Layout | Order Summary (Tab)
Rows Layout | Address (Tab)
Command Bar
How Do IGet There
Custom Control | Customer
Add
Popups

*Figure 89: Preparing a Spot to Draw a Bing Maps Control*

The rendering code adds two HTML DIV elements; one to display the Bing map and one to host the directions. It then instructs the widget to draw a Bing map and highlight the current customer's address.

```
myapp.AddEditCustomer.Customer_render = function (element, contentItem) {

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

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

    var directionsDiv = $("<div id='directions' class='msls-hauto msls-vauto'
></div>");

    directionsDiv.appendTo($(element));



    mapDiv.lightswitchBingMapsControl({

        street: contentItem.value.Street,

        city: contentItem.value.City,

        zipcode: contentItem.value.ZipCode,

        state: contentItem.value.Country,

        mapTypeId: Microsoft.Maps.MapTypeId.road,

        height: "470"

    });

};
```

Code Listing 45

Change the code behind the execute function of the HowDoIGetThere function to use jQuery to find the DIV element containing the map, and draw the route between the current location and the customer's address:

```
myapp.AddEditCustomer.HowDoIGetThere_execute = function (screen) {

    $("#addressMap").lightswitchBingMapsControl("getLocationOfUser", 
$("#directions"));

};
```

Code Listing 46

Save and refresh the browser to see that the add-edit customer screen indeed has a map showing the location of the current customer, as well as the how do I get there button. This button will display instructions on the map on how the end user can get from his or her current location to the particular customer.

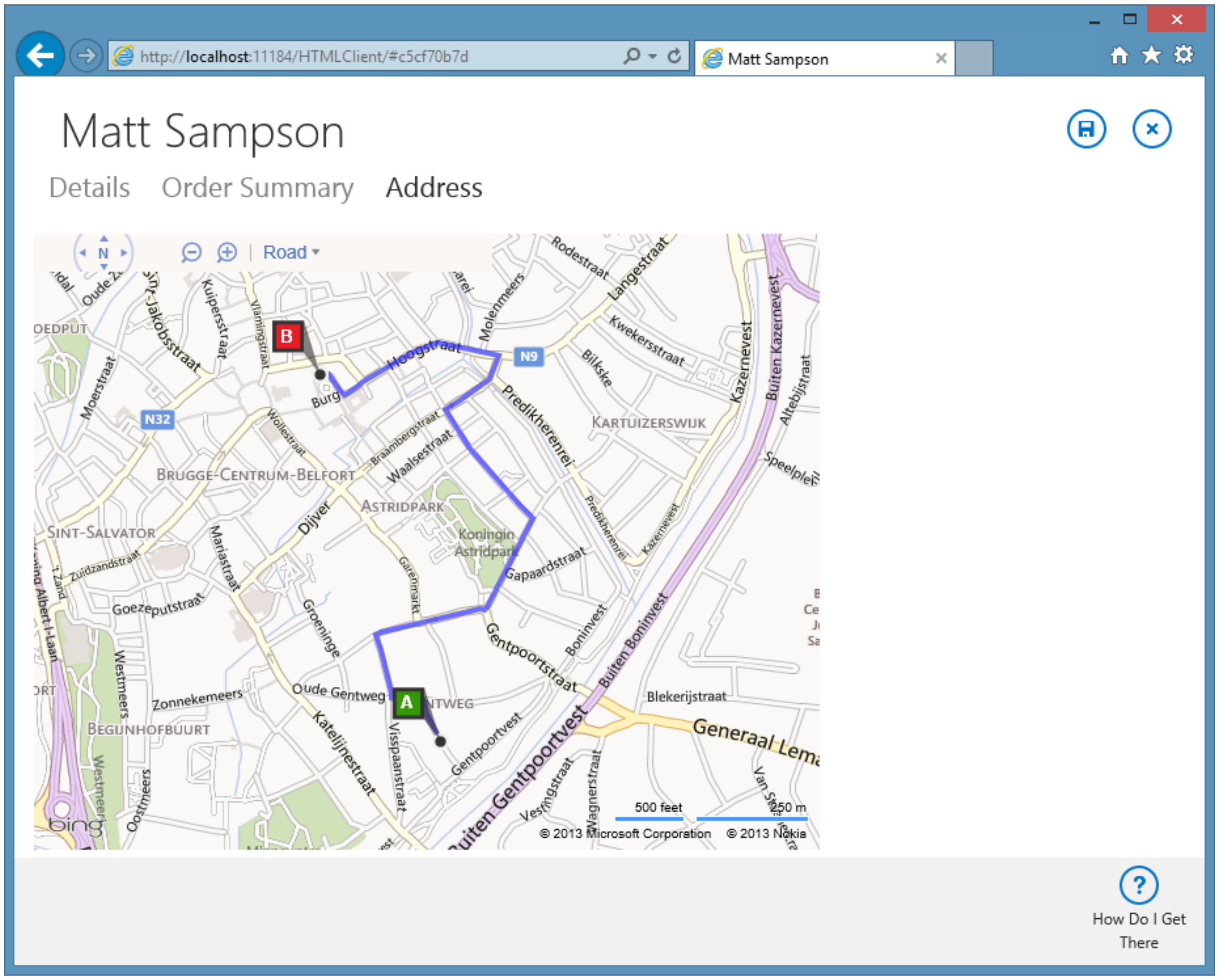


*Figure 90: The Bing Maps Control Showing Navigation Instructions to the Customer's Address*

# Chapter 6  Global styling and branding

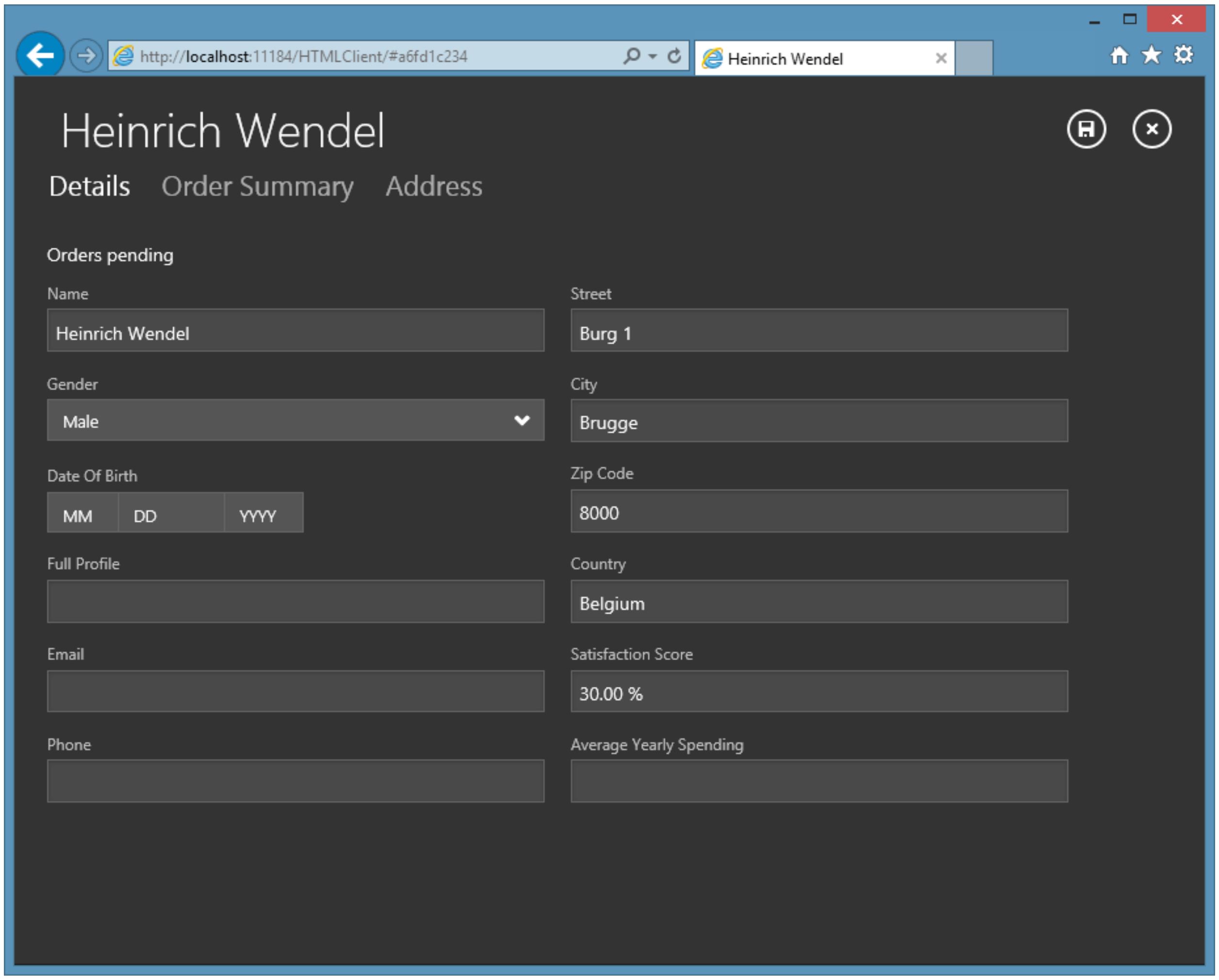
## Built-in schemas

LightSwitch HTML applications come with two global color and styling schemas: the default Light theme and Dark theme.

Switching between themes is done by opening the default.htm page again and editing the links to the following two stylesheets as explained in the preceding HTML comment.

```
    <!-- Change light-theme-2.0.0.CSS and msls-light-2.0.0.CSS to dark-theme-2.0.0.CSS and msls-dark-2.0.0.CSS respectively to use the

         dark theme.  Alternatively, you may replace light-theme-2.0.0.CSS with a custom jQuery Mobile theme. -->

    <link rel="stylesheet" type="text/CSS" href="Content/light-theme-2.0.0.CSS" />

    <link rel="stylesheet" type="text/CSS" href="Content/msls-light-2.0.0.CSS" />
```

*Code Listing 47*



Heinrich Wendel

Details Order Summary Address

Orders pending

| | |
|---|---|
| Name: Heinrich Wendel | Street: Burg 1 |
| Gender: Male | City: Brugge |
| Date Of Birth: MM DD YYYY | Zip Code: 8000 |
| Full Profile: | Country: Belgium |
| Email: | Satisfaction Score: 30.00 % |
| Phone: | Average Yearly Spending: |

*Figure 91: The Add Edit Customer Screen Using the Built-In Dark Theme*

## Custom jQuery Mobile themes

Did you notice this HTML comment?

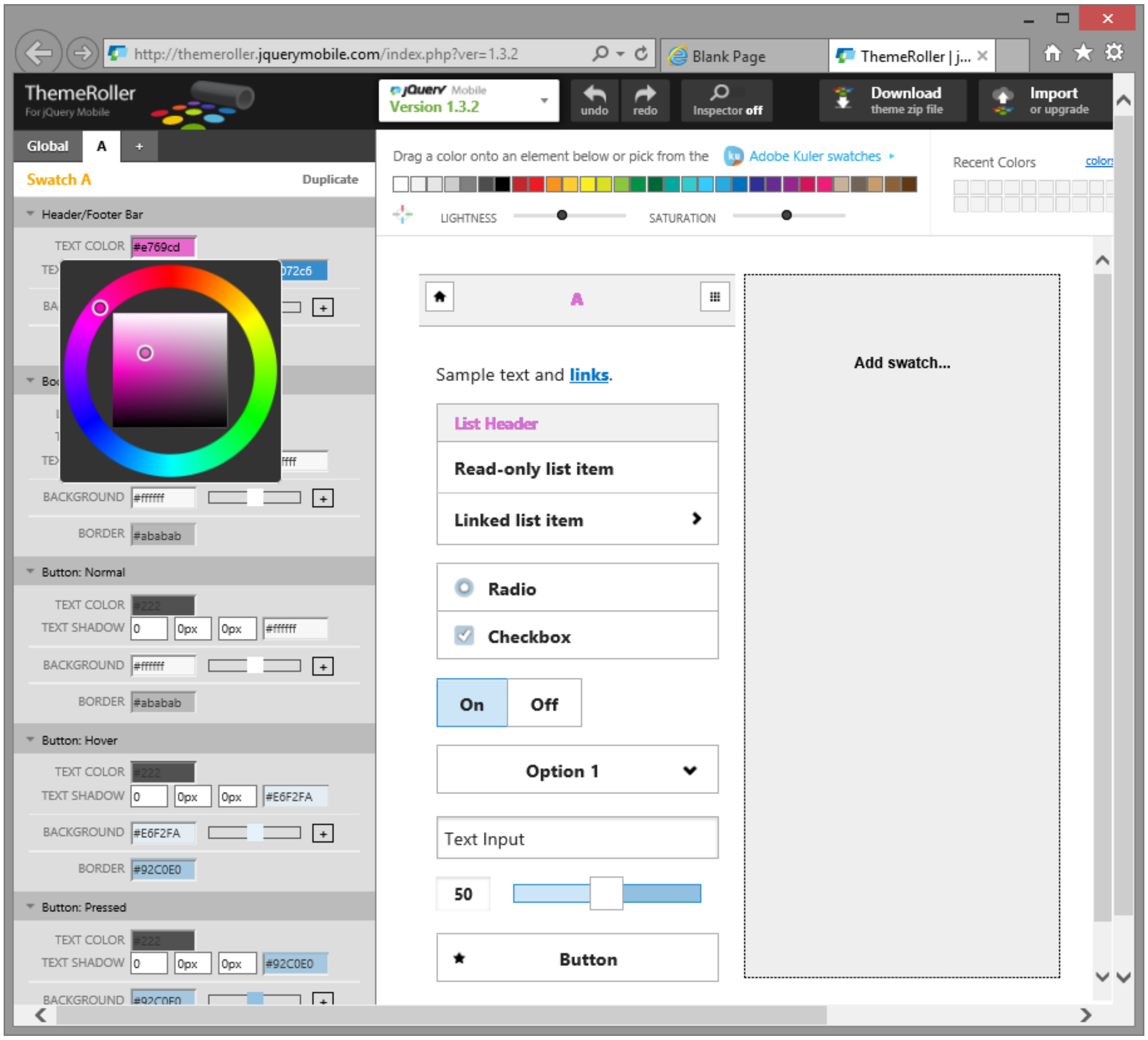
```
Alternatively, you may replace light-theme-2.0.0.css with a custom jQuery
Mobile theme.
```

*Code Listing 48*

LightSwitch uses an open source UI framework called jQuery Mobile. Relying on open source frameworks implies you get free bug fixes, upgrades, and awesome tooling, like jQuery Mobile ThemeRoller. ThemeRoller is a website where you can create custom themes that work with any jQuery Mobile application, thus allowing you to replace the built-in LightSwitch themes.

To create your own theme, simply copy the contents of light-theme-2.0.0.CSS from the Contents subfolder of the HTML project to your clipboard. Next, open http://jquerymobile.com/themeroller and find the Import button at the top. Paste the CSS into the dialog that appears, and the LightSwitch theme should be loaded into the ThemeRoller.

ThemeRoller exposes an overview of available properties on the left-hand side of the screen, which allows you to tweak every color, font, size, or rounding. In the center you will see an immediate preview reflecting each change you make as you adjust as much as you need until the style matches your corporate brand or desired theme.



Header/Footer Bar
TEXT COLOR #e769cd

Button: Normal
TEXT COLOR #222
TEXT SHADOW 0 0px 0px #ffffff
BACKGROUND #ffffff
BORDER #ababab

Button: Hover
TEXT COLOR #222
TEXT SHADOW 0 0px 0px #E6F2FA
BACKGROUND #E6F2FA
BORDER #92C0E0

Button: Pressed
TEXT COLOR #222
TEXT SHADOW 0 0px 0px #92C0E0
BACKGROUND #92C0E0

Figure 92: Creating a Custom JQuery Mobile Theme

Once you're satisfied, invert the import process by clicking the download button at the top of the contents folder of your project, and then referencing it from the default.htm page instead of the default light theme.

```
    <!--<link rel="stylesheet" type="text/css" href="Content/light-theme-
2.0.0.css" />-->

    <link rel="stylesheet" type="text/css" href="Content/Syncfusion-
blue.min.css" />
```

*Code Listing 49*

Creating a custom jQuery Mobile theme, and thus a LightSwitch theme, can be done in a few minutes. However, the art of designing a good theme is well beyond the scope of this book (and the skill set of the author).

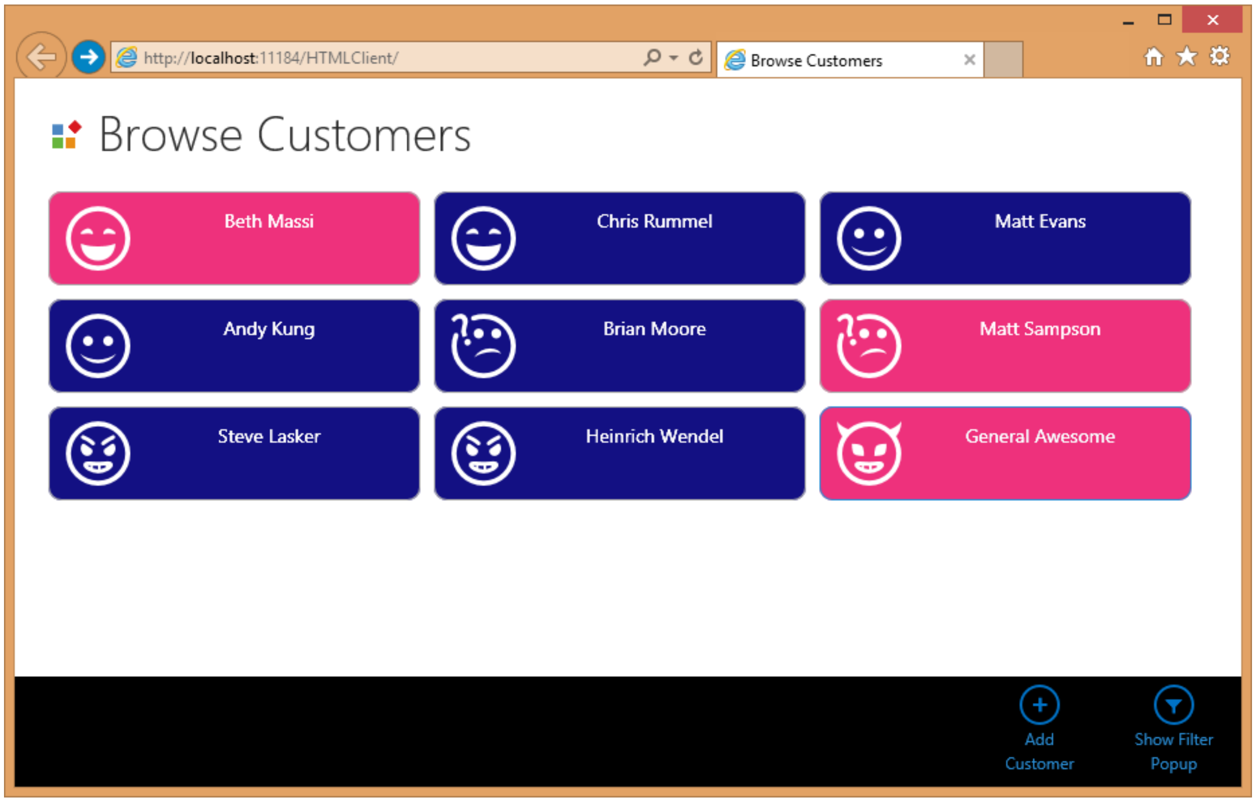


*Figure 93: Application with Black and Blue Theme*

## Adding a company logo

LightSwitch HTML applications use two logos, which can both be found in the images subfolder of the content folder of your HTML client project: user-splash-screen.png and user-logo.png. The former will be visible while the application is loaded; the latter is displayed at the top left of every browse screen.

Both images can and should be replaced with your real company logo by simply changing the resource itself.

Additionally, I find that the footer (the command bar at the bottom) takes up too much space if you only put a couple of commands on it, so I often tweak it to include a fancy logo.

To do this, open up the user-customisations.CSS file again and append the following CSS:

```css
.msls-footer {

    background-image: url(http://www.syncfusion.com/Content/en-US/Home/Images/syncfusion-logo.png);

    background-repeat: no-repeat;

    background-size: contain;

}
```

*Code Listing 50*

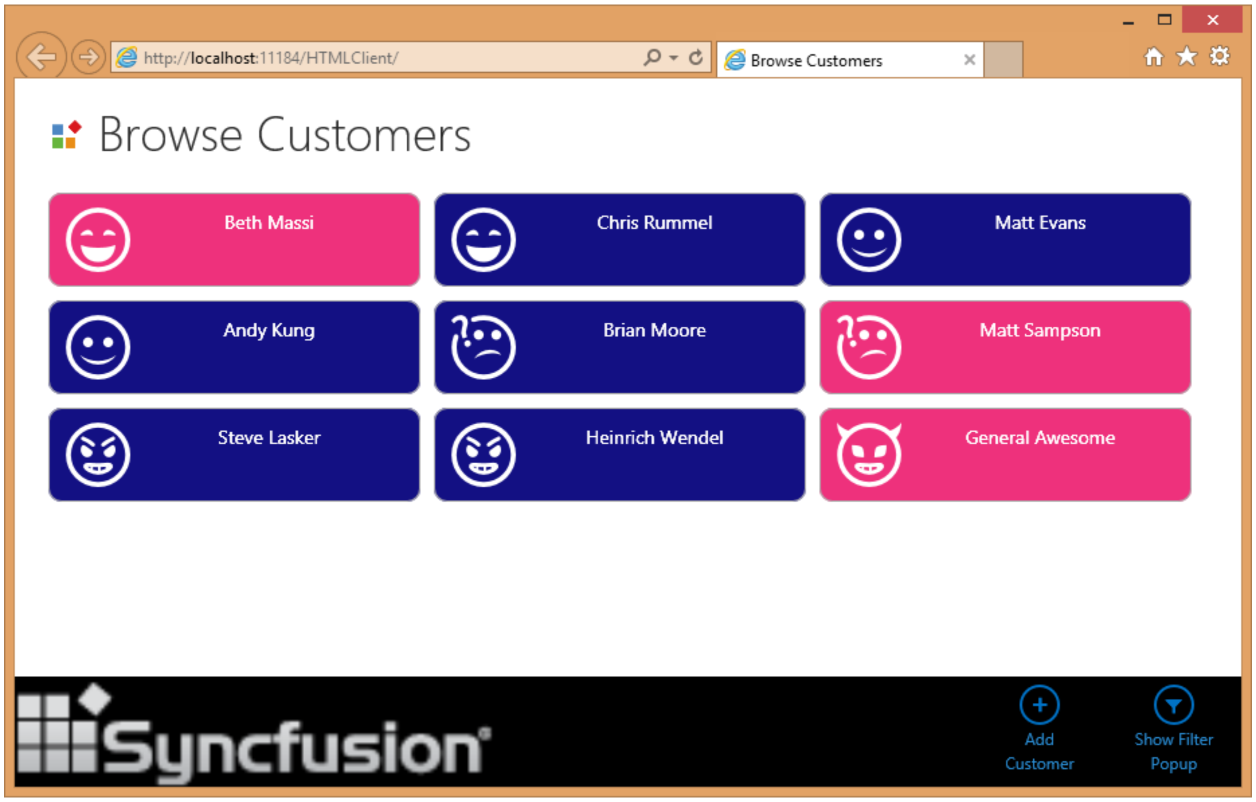


*Figure 94: Application with Custom user-logo.png and Additional Footer Styling*

# Chapter 7 Publishing

## Available hosting options

The only way software can deliver additional value to an end-user is in the form of shippable, runnable features. I have yet to meet an end user who became excited over an sln file (a Visual Studio solution), so let’s explore how to deploy a LightSwitch project to a production environment.

LightSwitch offers a variety of supported hosting environments out of the box, both on premise or in the Cloud. Your intrinsic database, the database that is automatically created based on your entities unless you chose to connect to an existing database, can be deployed to a SQL 2008, 2012, 2013, or SQL Azure server, and your HTML project and middle-tier can be hosted on either an IIS, on Azure Cloud Services, or, if you need less configuration, an Azure Web Site.

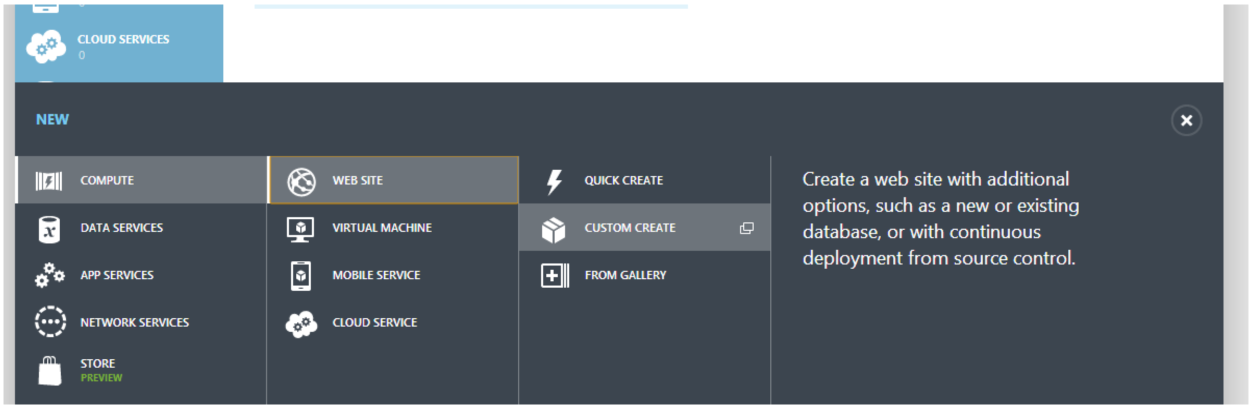
Alternatively, LightSwitch allows you to create Cloud Business Apps as well. This allows the app to be auto-hosted by a Sharepoint 2013 server either on premise or on an external provider like Office 365.

For now, you’ll stick with the easiest solution: deploying the application as a free Azure Web Site.

## Creating the Azure Web Site

To create an Azure Web Site, log in to your Azure account dashboard at [https://manage.windowsazure.com/](https://manage.windowsazure.com/).

Select web sites from the overview panel on the left-hand side, then click New at the bottom. From the menu that appears, select to custom create a new web site.



Create a web site with additional options, such as a new or existing database, or with continuous deployment from source control.

*Figure 95: Azure Dashboard Menu*

A modal dialog will appear to configure additional information about your new web site, like the name affecting the URL of the site, the region to the data center of your choice (the closer to the end users, the better), and options to create a new database.

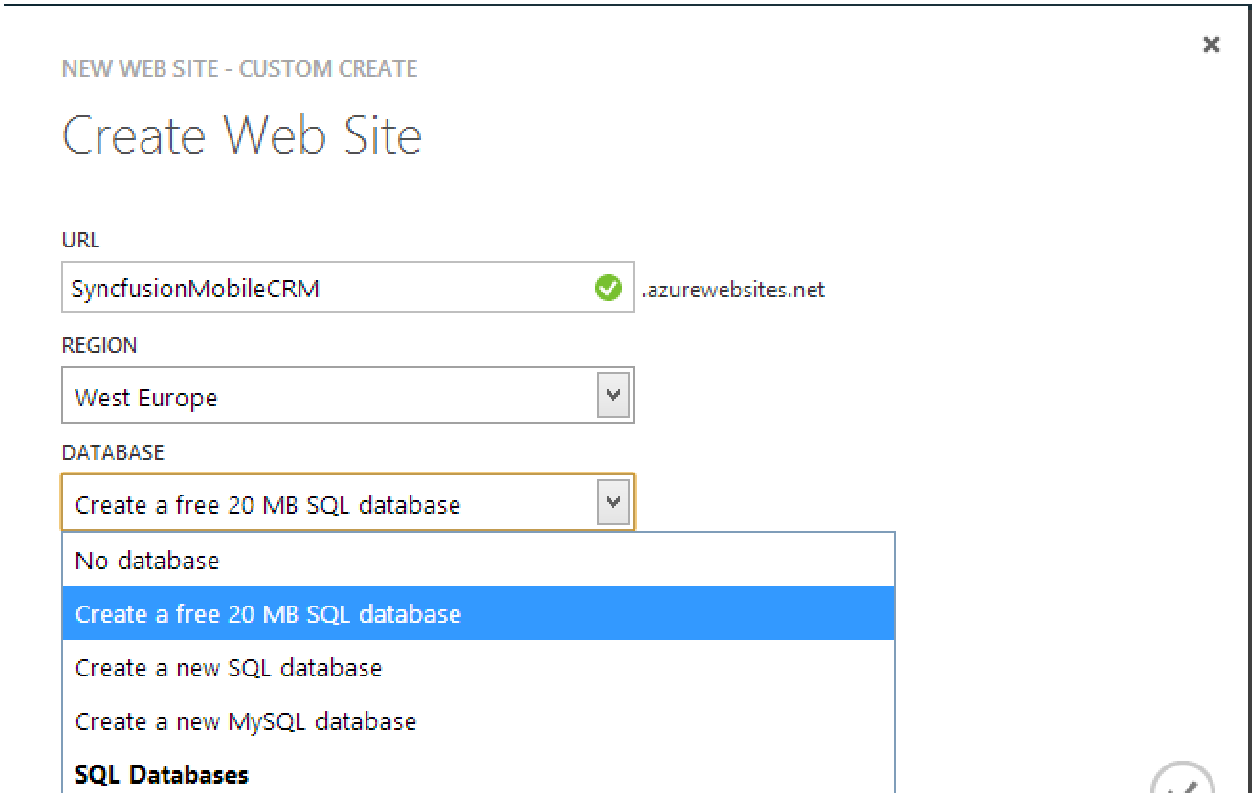

NEW WEB SITE - CUSTOM CREATE
Create Web Site
URL
SyncfusionMobileCRM
.azurewebsites.net
REGION
West Europe
DATABASE
Create a free 20 MB SQL database
No database
Create a free 20 MB SQL database
Create a new SQL database
Create a new MySQL database
SQL Databases

*Figure 96: Create Web Site Dialog*

In a second step, you'll be asked to name the database and select a region for it. Select the same data center as before. After a very brief pause, your web site will be ready.

## Publishing the application

With the supporting Azure infrastructure in place, you should be less than a minute away from having your first mobile business application ready to show off.

Change your build configuration from debug to release, then right-click on the LightSwitch project in the solution explorer. Select Publish from the context menu.

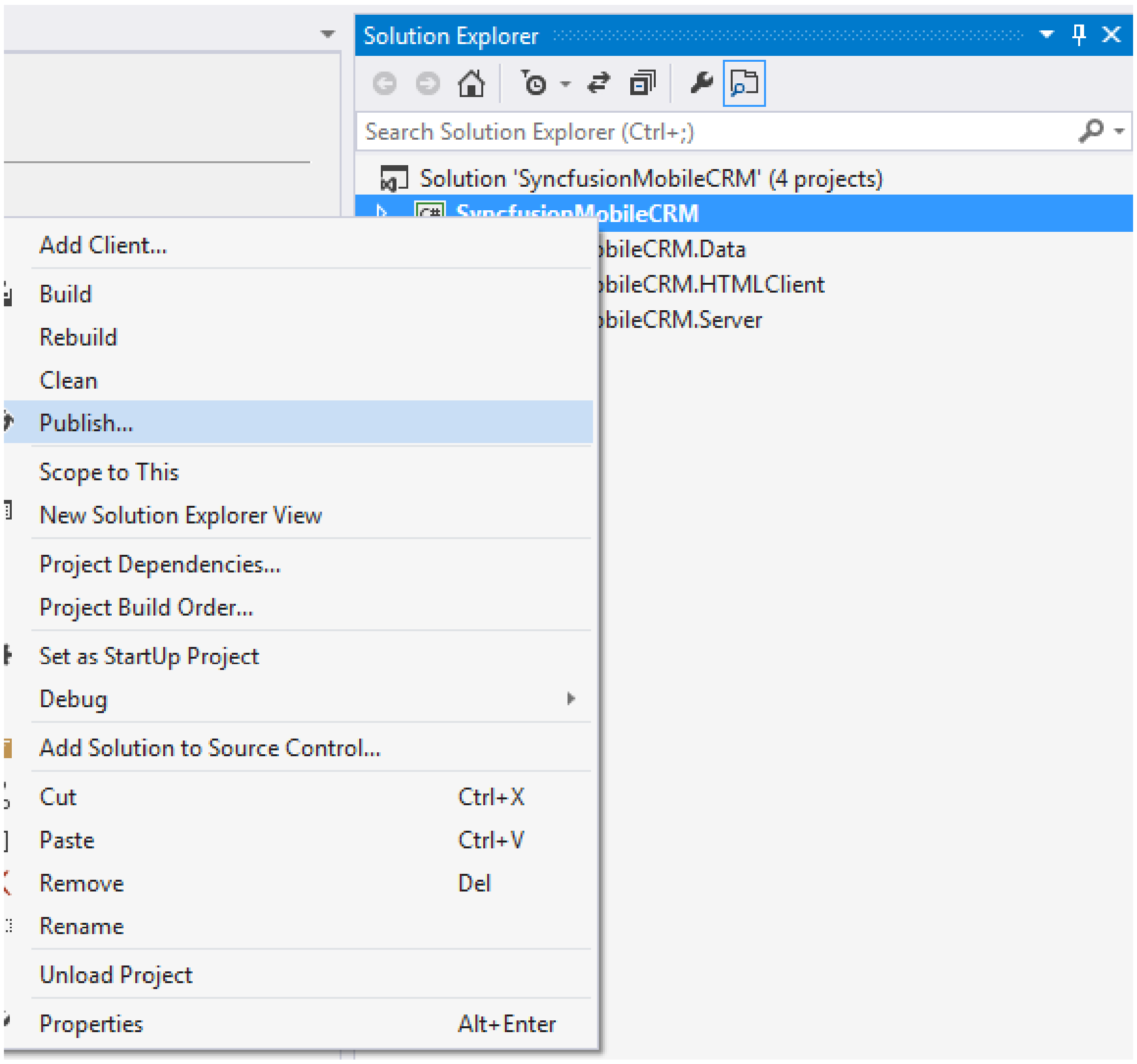
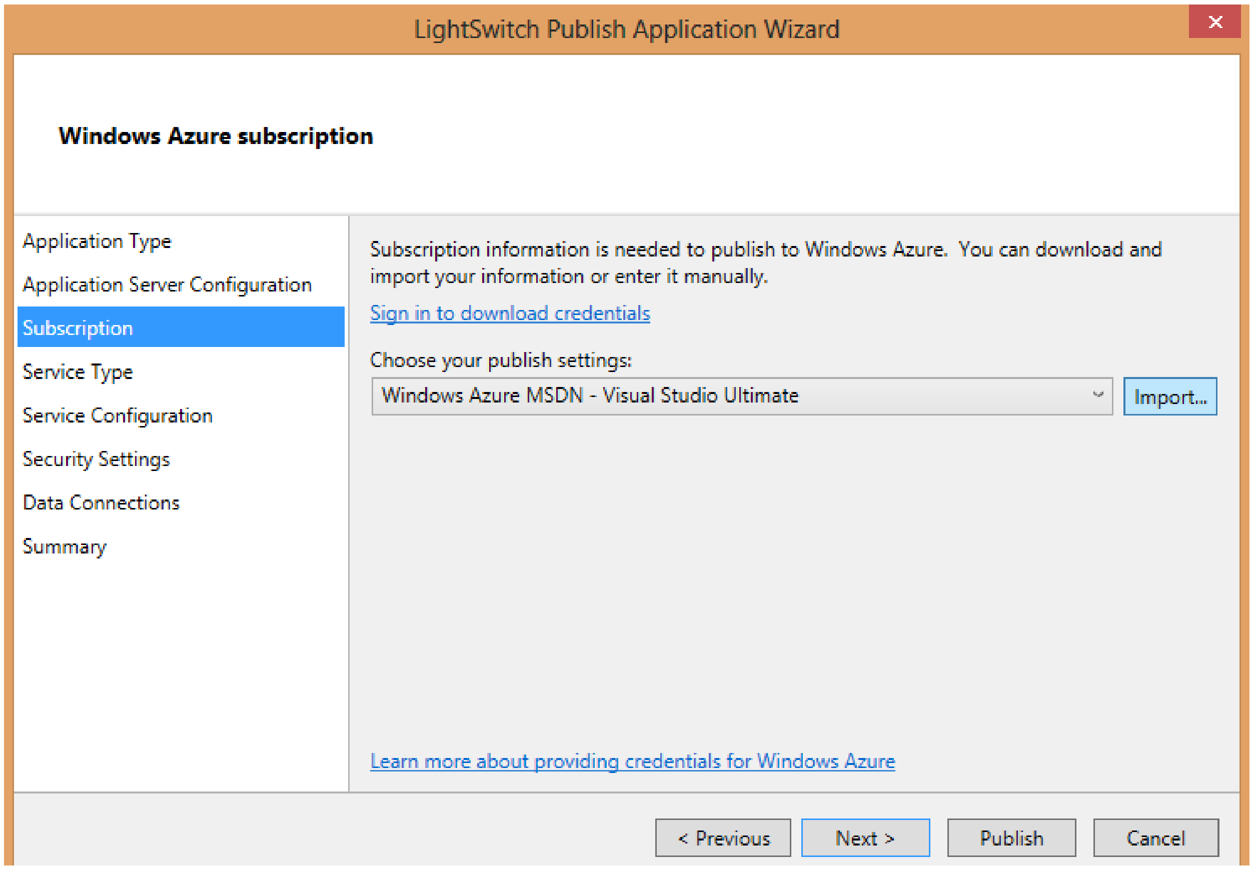


*Figure 97: Publish the LightSwitch application.*

This will open the LightSwitch publish application wizard to guide you through the process.

This wizard allows you to publish the application as a whole or deploy the middle tier as an OData service only. The second tab, application server configuration, allows you to choose either an on-premise installation to IIS or a deployment to Azure, which is of course the option you'll want to select. If you do not have the Azure SDK installed, Visual Studio will now guide you through the required installation.

The third tab, Subscription, is probably the biggest time saver.



Subscription information is needed to publish to Windows Azure. You can download and import your information or enter it manually.

Sign in to download credentials

Choose your publish settings: Windows Azure MSDN - Visual Studio Ultimate

Learn more about providing credentials for Windows Azure

*Figure 98: The Subscription Tab*

This tab allows you to connect to your Azure account from within Visual Studio to download technical information about your Azure account instead of manually having to configure everything. First, click Sign in to download settings. This will open your browser and, after you confirm your Azure credentials, start to download a settings file for you. From the wizard, you can now click the import tab to select that settings file.

With this in place, the rest of the wizard is simple. The name you gave to your web site will be available to select from the service configuration tab, and the connection to the database you created will already be filled in the data connections tab.

After reviewing the summary page for potential errors, click Publish. Visual Studio will build, package, upload, and deploy your application. Your post deployment scripts in the database project will be executed as well, providing you with the same sample data.

# Ready, set, go

Grab the nearest mobile device and browse to http://<yourAzureWebsite>.AzureWebsites.net/HTMLClient/default.htm (the default.htm can be omitted, but don't forget the HTMLClient subfolder) to view your first, mobile-oriented, touch-friendly, in-the-cloud, line-of-business, single-page application in action.

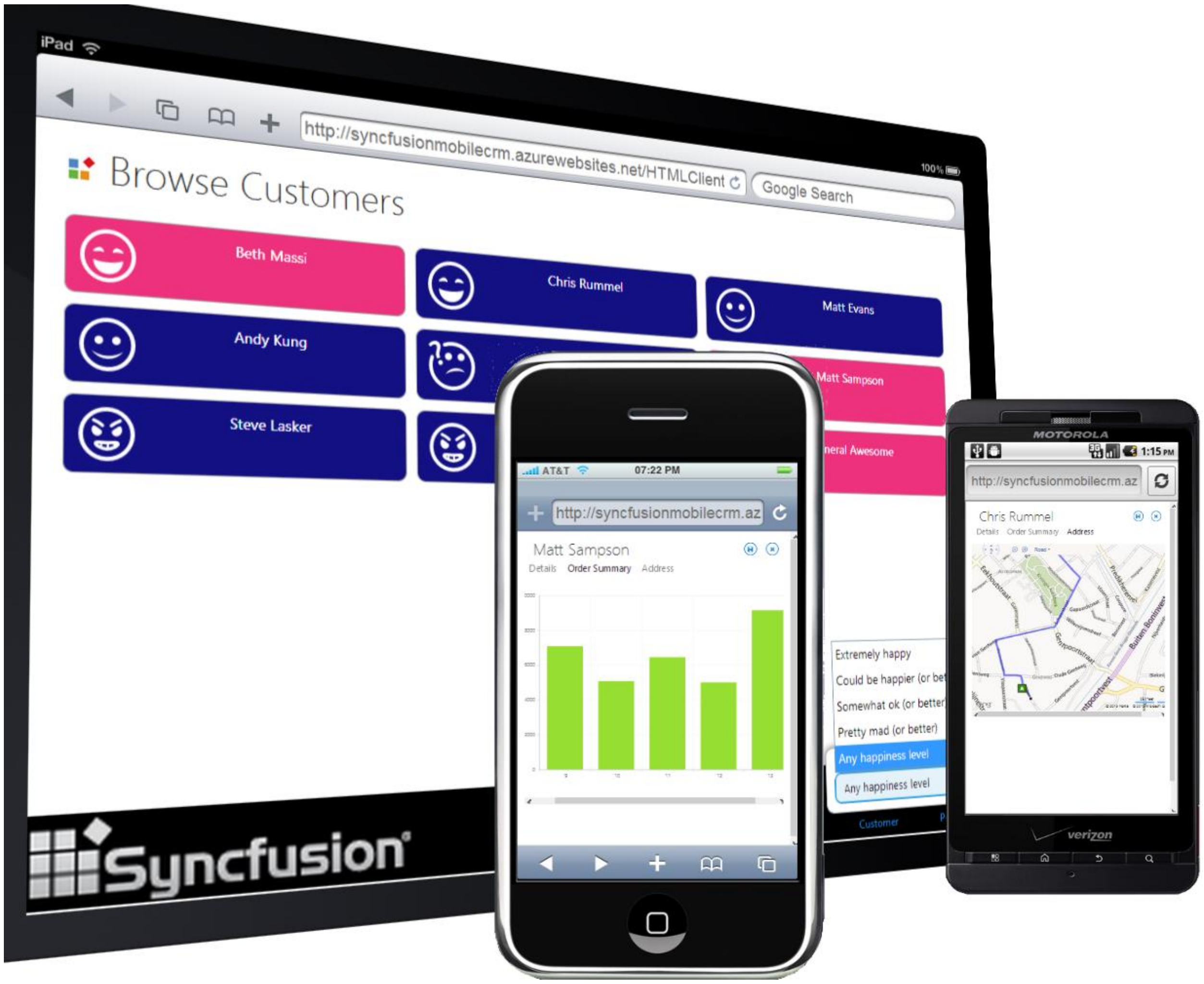


Figure 99: A Published Application

Congratulations! You've survived your first day at the LightSwitch mobile business app academy.

You'll have to figure out the rest of the process yourself from here, but there are a lot of resources to help you. For example, you can browse the official LightSwitch developer center (http://msdn.microsoft.com/en-us/vstudio/htmlclient.aspx) for tons of great learning material, or hang out at the largest unofficial LightSwitch community (http://lightswitchhelpwebsite.com/) where LightSwitch rock star Michael Washington treats us with incredible content at an amazing pace. Speaking of incredible content, you should also check out the blogs of pragmatic Paul Van Bladel (http://blog.pragmaswitch.com/), artist Jewel Lambert (http://jewellambert.com/), or the wizards at Xpert360 (http://xpert360.com/).

I'm sure I'm leaving out at least a dozen great blogs, including the blog of Alessandro Del Sole (http://community.visual-basic.it), who did a ton of work technically editing this e-book, and my own ramblings (http://janvanderhaegen.wordpress.com). The best way to make sure you don't miss a thing is to download the community app (http://apps.microsoft.com/windows/en-us/app/visual-studio-lightswitch/d5d5f80e-d8df-4fa9-935b-a92ed7ad03a7) and subscribe to the LightSwitch team blog (http://blogs.msdn.com/b/lightswitch/) where you'll be treated to a thorough monthly rollup of content and community by Senior Program Manager Beth Massi.

Thanks to everyone for keeping the LightSwitch community informed and engaged. Keep rocking LS!