



LightSwitch

Succinctly

by Jan Van der Haegen

LightSwitch Succinctly

By
Jan Van der Haegen

Foreword by Daniel Jebaraj



Copyright © 2012 by Syncfusion, Inc.

2501 Aerial Center Parkway

Suite 200

Morrisville, NC 27560

USA

All rights reserved.

I mportant 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, 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.

E dited by

This publication was edited by Praveen Ramesh, director of development, Syncfusion, Inc.

Table of Contents

About the Author	8
Introduction	9
Preface	10
Chapter 1: Where Do I Get Visual Studio LightSwitch 2012?	12
Chapter 2: Hello World.....	13
Create a new project.....	13
Always start with data	13
Add some screens	15
Press F5 to blend it all together	16
Chapter 3: The Entity Designer	18
Creating a Task entity with simple and computed properties.....	18
Creating a Person entity with clever reuse of existing business types	23
Taking full control.....	24
Don't forget to design the relationships.....	25
Data-in	29
To OData or not to OData	30
Data-out	33
Blending data-in and data-out.....	34
Chapter 4: The Query Editor	36
Finding the tasks that matter.....	36
Adding filter criteria	37
Adding some logical sorting	38
Add a screen directly to this query	39
Finding the important tasks per user	40
Query inheritance.....	40
Adding parameters to a query.....	41
Chapter 5: The Screen Editor.....	43
Creating a Search Data Screen for our query	43

The Model and ViewModel part of the Screen Editor	43
Turning a Screen Property into a required Screen Parameter	44
Validation on Screen Properties.....	45
Initializing Screen Properties.....	46
Where did the screen go?	47
Adding a new command to a screen	47
Opening a screen from code.....	49
The run-time Screen Editor	51
Hiding the <i>TaskAssignedPerson</i> controls	52
Creating new item templates.....	53
Modifying an entire group of controls at once	55
Chapter 6: Application Editor.....	57
Extending LightSwitch applications.....	57
Installing an extension in Visual Studio	57
Activating an extension in your application	58
The General Properties tab.....	60
Screen Navigation.....	61
The Access Control tab.....	63
The Application Type tab	65
Publishing time... ..	66
Chapter 7: Moving On	67
Is Visual Studio LightSwitch the right tool?	67
Learning more about Visual Studio LightSwitch.....	67

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.

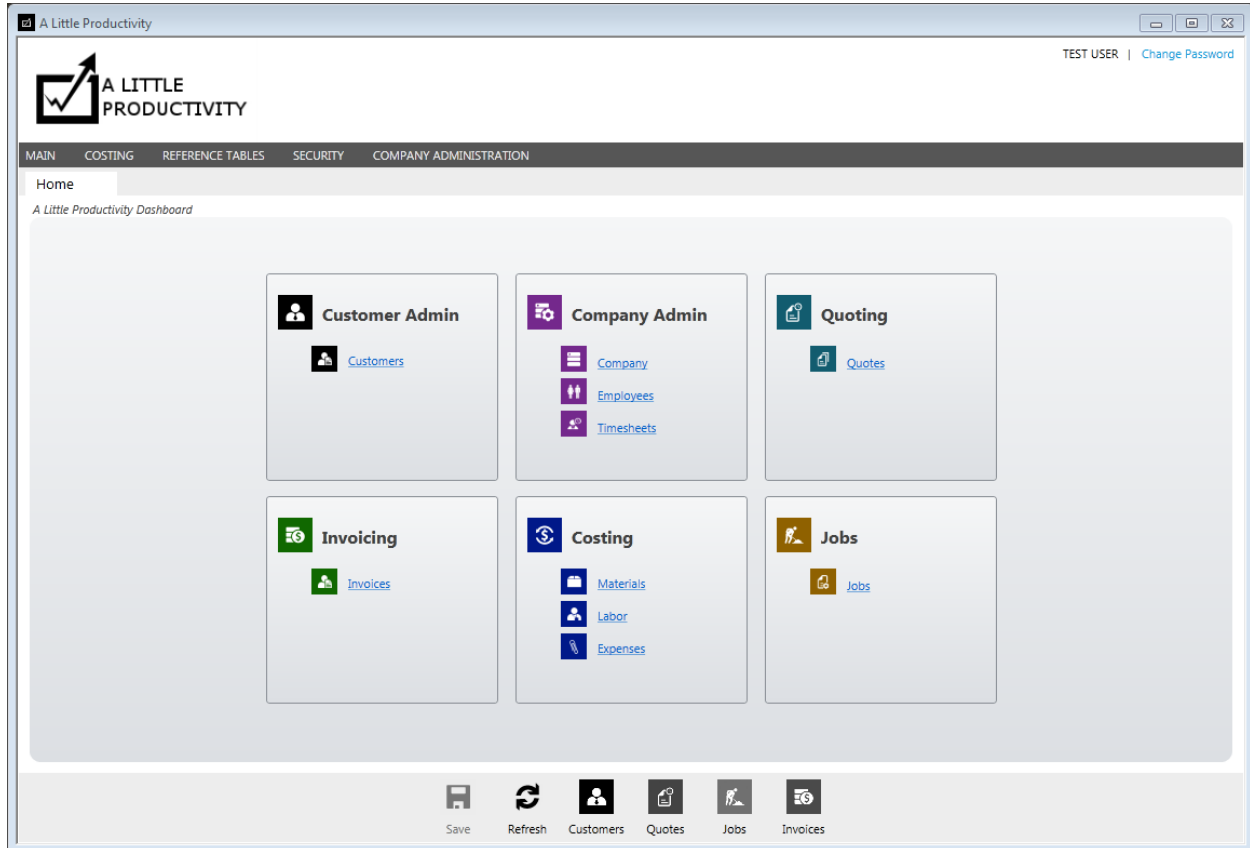
Please follow us on Twitter and “Like” us on Facebook to help us spread the word about the *Succinctly* series!



About the Author

Jan Van der Haegen is a green geek who turns coffee into software. He's a loving husband, proud to be part of an international team at Centric—one of the top 5 IT suppliers in Belgium—and so addicted to learning about any .NET technology—Visual Studio LightSwitch in particular—that he maintains a blog on his coding experiments, writes a monthly LightSwitch column for MSDN magazine, and recently founded switchtory, a LightSwitch-oriented software development and consultancy start-up. You can find his latest adventures at <http://www.switchtory.com/janvan>.

Introduction: Exploring Visual Studio LightSwitch 2012 (RC)



Time for a Little Productivity...

In mid-2011, Microsoft introduced Visual Studio LightSwitch as “the easiest way to build data-centric applications for the desktop or the cloud.” As appealing as this slogan might sound, a significant majority of IT-minded people still either don’t know about it, don’t know where to start, or have prejudiciously categorized it as “a tool for Microsoft Office users.”

However, you are likely aware of the fact that choosing the right tool for each challenge is getting half of the job done already. Whether you have never written a software application yourself before, or you are a professional software engineer, with this e-book, Syncfusion is offering you the opportunity to take a small amount of time to get to know Visual Studio LightSwitch 2012 (RC) and find out if and how this product could fit in your tool belt. Instead of a complete “getting started” tutorial, this e-book takes you for a sight-seeing tour throughout the LightSwitch development environment to teach you as much about the product in as little time possible. This walk shouldn’t take much more than an hour to complete, and if LightSwitch turns out to be a tool suitable for your needs, you’ll win back that hour and many more before the working day is over, you have my personal guarantee.

Preface: Why Would Exploring LightSwitch be Worth this Hour of My Time?

Small minorities of us design realistic 3-D games, or are on Google's Project Glass team. Most of us in the IT industry create very specific line-of-business software that we'll never be using ourselves. Although the purpose and specifics might change from industry to industry, software we write for maintaining medical appointments, accounting for businesses, keeping track of college students, or managing real estate property, all fall in the same type of boring software called "information systems." Information systems are data-centric applications that have several things in common: One or multiple databases, loads of business rules that define how that data can be modified, one or more different graphical user interfaces, and lots of different reports.

However unchallenging these information systems might be, the opposite is true of the people who create them. Having spent months or even years to fully get to know the coding language, technologies, and patterns, and often armed with a computer science degree, we are analytical-minded artists always looking for challenge.

Michael A. Jackson identified this paradox in 1975 in his book *Principles of Program Design*: "Programmers [...] often take refuge in an understandable, but disastrous inclination toward complexity and ingenuity in their work. Forbidden to design anything larger than a program, they respond by making that program intricate enough to challenge their professional skill." This still holds true, 37 years later, and as a direct result software is still ridiculously expensive, almost never delivered on time, full of known issues, and ironically enough, although created to aid specific users in a specific business case, often needs a 500-page manual or a couple of days of training to teach users how the software will help in their own business process.

The most successful software engineers or software companies are the ones that understand how writing software should be boring. They're the ones that understand there's an ever-evolving, revolutionary wealth of technological choices that could be made, but that these choices aren't the core of their business. They are the true rock stars that can deliver software before deadlines and under budget that does exactly what the business needs. There are really only a handful of rules to adhere to in order to follow in their footsteps toward success:

- **Learn the business.** It's preposterous to believe you can create software to solve someone's problem, or make someone's life easier, if you do not know who that someone is, and what his or her problem is. Before starting development, ask the end user to take you through his or her day. With every move, ask "Why?" "How?" and most importantly, "How does that make you feel?" Don't expect the end user to be able to provide you a list of requirements—the fish are the last to discover the ocean, as the Chinese proverb goes. Be an insider to the business, but enough of an outsider to identify what the actual needs are.
- **Serve only the business.** There are 24 hours in a day, more than enough to learn new, challenging technologies and architectural designs or code patterns, but do that in your leisure time, not on the job. Spending time on something that doesn't directly add value for the business is considered waste. Expensive waste as well. If the users need

software to manage their appointments, then that's what they should get—not a “generic plugin-based macro recorder,” or whatever you tricked yourself into believing will add true enlightenment to their jobs.

- **Code only the business.** Tedium is inescapable; no auto-mapper or code generator can negate the fact that screens, entities, and validation have to be defined. However, define it once. If the overwhelming majority of your hand-written code isn't domain specific, or if you are implementing changes in several layers throughout your application, you are doing it wrong. More specifically, you are using the wrong tool.

If you read this preface, and you agree, or maybe have even been nodding or smiling, then Visual Studio LightSwitch is a tool you'll definitely want in your tool belt. Designed for citizen developers and professional software engineers alike, if you are facing an information system, if you know your business and are ready to serve the business, then Visual Studio LightSwitch will allow you to code only the business and become the coding rock star you deserve to be.

Chapter 1 Where Do I Get Visual Studio LightSwitch 2012?

Acquiring LightSwitch 2012 (RC) is the easy part. At the time of writing this book, it is available only as a first-class citizen in Visual Studio 2012, which is still in Release Candidate phase and can be [downloaded for free](#). When the RC period of Visual Studio is over, the RTM version will integrate LightSwitch as well, so you're safe if you have an MSDN subscription or otherwise intend to purchase a license for Visual Studio Professional or above.

LightSwitch 1.0, called LightSwitch 2011 at the time, was initially released as an out-of-band release. Because of this, it was also made available as a separate purchase mainly targeting non-professional developers, costing about \$199 per developer. In this stand-alone version, the Visual Studio shell is installed and some of the more advanced LightSwitch extensibility features are unavailable. According to the announced road map, this option will not be repeated for Visual Studio LightSwitch 2012.

Chapter 2 Hello World

If you're anything like me, you have some things you'd like to do, a lot of projects you have to do, and tons of tasks you should have already done. As part of this tutorial, we'll be writing a simple LightSwitch application to help you organize people, tasks, and projects. After all, *time is money*.

Create a new project

Once Visual Studio 2012 RC is installed, start it up and from its **File** menu, select **New Project**. In the installed templates, choose the option to create a new **LightSwitch Application (Visual C#)**. You could opt to create your application using VB.NET as the coding language as well. Give your project a suitable name and click **OK**.

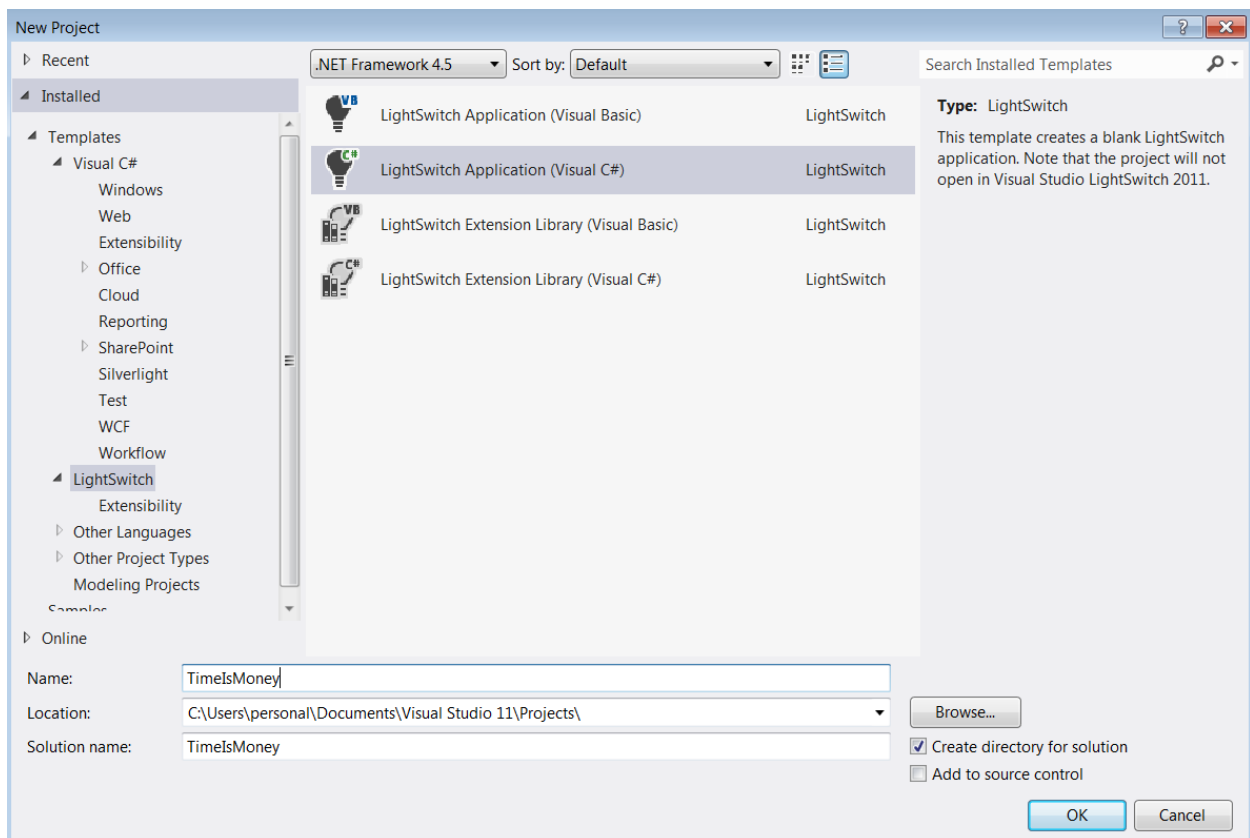


Figure 1: Creating a new LightSwitch project

Always start with data

The LightSwitch home page is shown, which suggests to start with data—an obvious suggestion given the data-centric nature of the applications you'll find that LightSwitch is a good fit for. Create a new entity by clicking the **Create New Table** link. This brings up the Entity Designer.

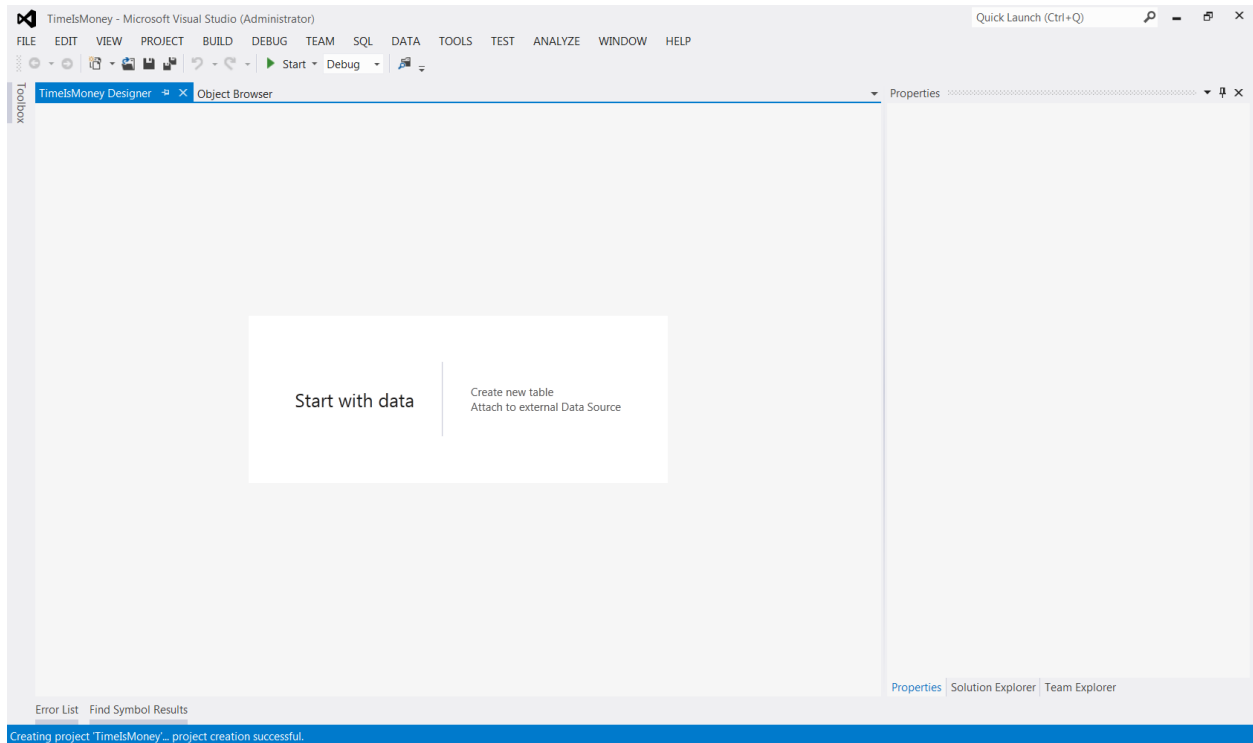


Figure 2: LightSwitch home screen

We'll explain everything that goes on in the Entity Designer throughout the book. First, let's make a working "hello world" application by renaming the entity to **Project** and adding one string property called **Name**.

To rename the entity, double-click on it in the Entity Designer or fill out the **Properties** window on the right.

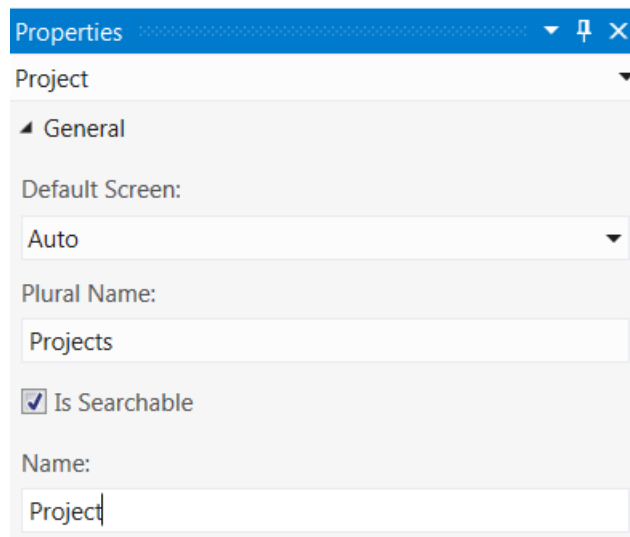


Figure 3: Renaming the entity


Project			
Name	Type	Required	
 Id	Integer	<input checked="" type="checkbox"/>	
Name	String	<input checked="" type="checkbox"/>	
<Add Property>		<input type="checkbox"/>	

Figure 4: Adding the Name string property

Add some screens

Just above the Entity Designer, you'll find some possible things to add to, or do with this entity. The **Write Code** button in the top right (as shown in Figure 5) is one that you'll encounter in almost every designer you use in LightSwitch, and is probably what makes LightSwitch so powerful. LightSwitch is not a black box; you can take control of every part using plain old .NET code—VB.NET or C#—using these well-defined extension points.

Click the **Screen...** button to add a new screen as highlighted in the following figure.

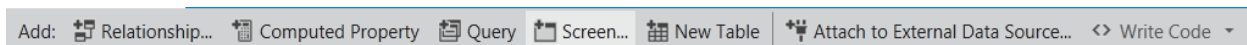


Figure 5: Toolbar in the Entity Designer

A pop-up will appear offering you several screen templates. Personally, I almost always start with the **List and Details Screen** template.

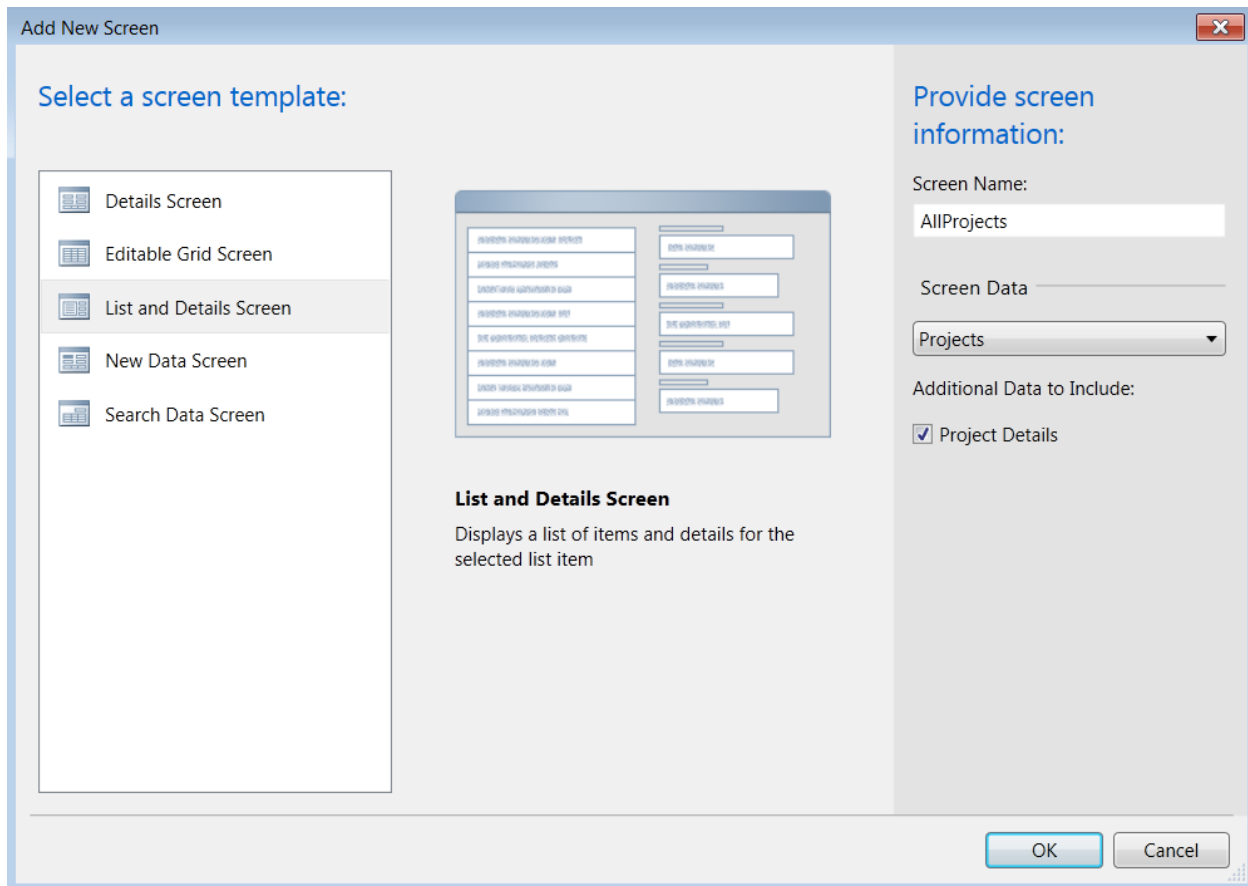


Figure 6: Add New Screen wizard

Provide a suitable screen name—*AllProjects*, for example—and select our project entities as the screen data to use. Make sure you select the **Additional Data to Include: Project Details** check box before clicking **OK**. LightSwitch then automatically uses the first string property as the summary for the entity and shows the value of all the properties in the details view.

Press F5 to blend it all together

Press **F5** or select **Debug > Start Debugging** from the menu to build and launch your application. Building a LightSwitch application can take quite some time, but after this small delay your application will start and be ready for use.

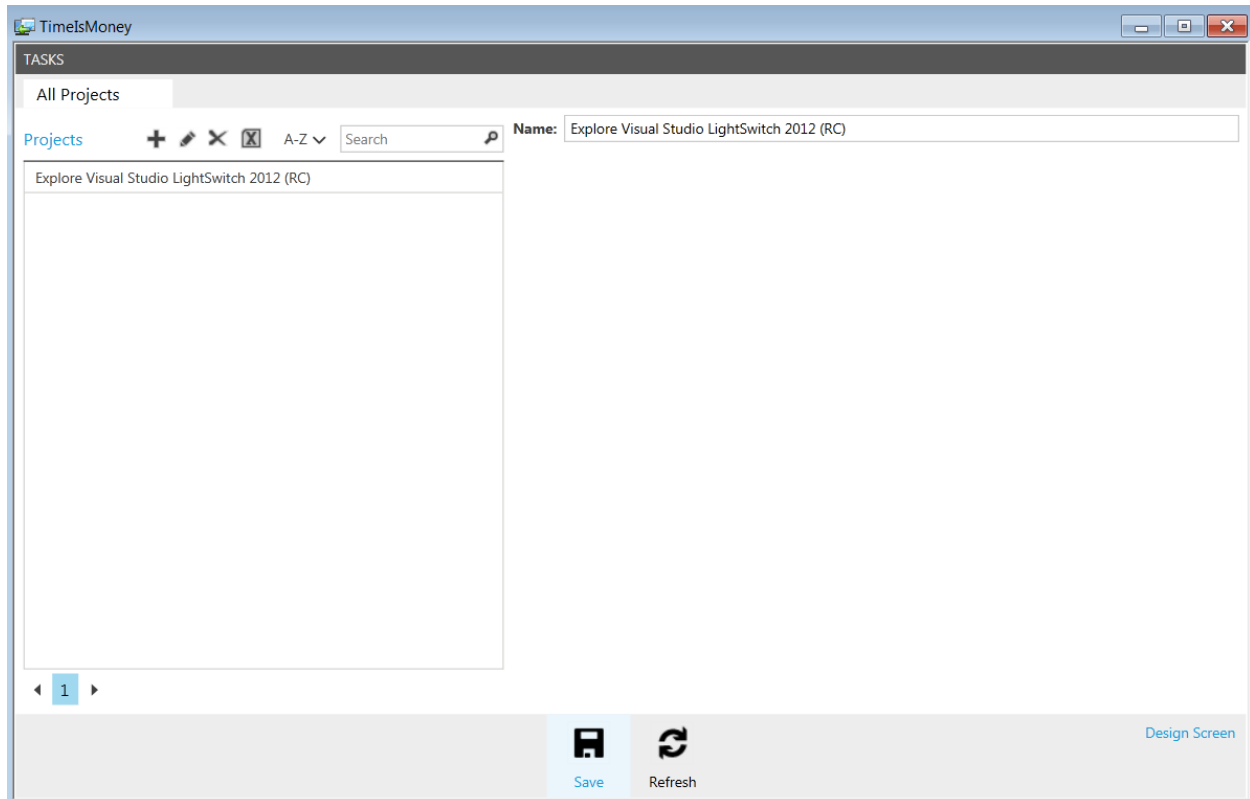


Figure 7: Hello World application

At this point, we have a working application in which we can add, edit, or delete projects. Although we haven't written any code because there is no domain logic (the only code you'll have to write) yet, the application has a ton of functional and nonfunctional features: collapsible menus, data paging, sorting, searching, exporting data to Excel, validation, concurrency management, and more.

Chapter 3 The Entity Designer

Creating a Task entity with simple and computed properties

In the **Solution Explorer**, right-click on **Application Data** and select **Add New Table**.

Our second entity will be called *Task*, and it will have a couple of properties.

Task			
Name	Type	Required	
Id	Integer	<input checked="" type="checkbox"/>	
Name	String	<input checked="" type="checkbox"/>	
Description	String	<input checked="" type="checkbox"/>	
DueDate	Date	<input checked="" type="checkbox"/>	
Priority	Integer	<input checked="" type="checkbox"/>	
PercentageComplete	Percent	<input checked="" type="checkbox"/>	
State	String	<input type="checkbox"/>	
Summary	String	<input type="checkbox"/>	
<Add Property>		<input type="checkbox"/>	

Figure 8: Task entity in the Entity Designer

For each of the properties, the Entity Designer shows three columns: the name of the property, the type, and a check box that indicates if a value is required.

When you select a particular property, additional extended properties can be edited in the **Properties** window where you can take more fine-grained control over how the entity or its properties are visualized, stored, validated, etc.

For example, I removed the maximum length of the **Description** property as shown in the following figure.

Validation

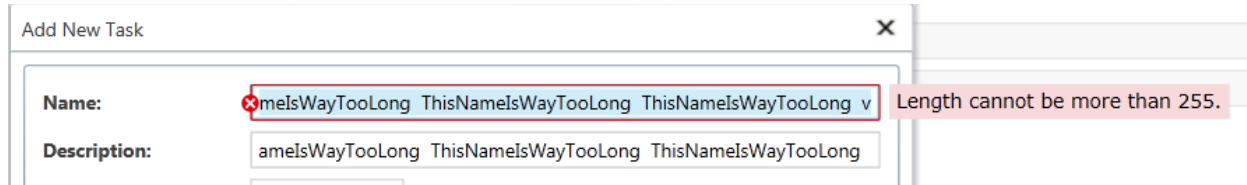
☒ Is Required

Maximum Length:

[Custom Validation](#)

Figure 9: Clearing the maximum length

Specifying the Maximum Length for Description field here is a great example of the “Don’t Repeat Yourself” (DRY) principle that I implicitly referred to in the preface. In the entire application, there is only one source of truth that controls how long the **Description** property can be. This information will be used to create the actual database fields correctly, but also to implement validation on the server and the client.



The screenshot shows a web form titled "Add New Task". It has two input fields: "Name:" and "Description:". The "Name:" field contains the text "meIsWayTooLong ThisNameIsWayTooLong ThisNameIsWayTooLong v" and is highlighted with a red border. A red error message "Length cannot be more than 255." is displayed to the right of the field. The "Description:" field contains the text "ameIsWayTooLong ThisNameIsWayTooLong ThisNameIsWayTooLong".

Figure 10: Length validation is still active for the **Name** property

Similarly, I limited the values of the **PercentageComplete** property to fall between 0 and 100 (inclusive).



The screenshot shows a configuration panel for "Percent Decimal Places". It has a text input field with the value "0". Below it is a checkbox labeled "Display by Default" which is checked. There is a section titled "Validation" with a checkbox labeled "Is Required" which is checked. Below that are two text input fields: "Minimum Value:" with the value "0" and "Maximum Value:" with the value "100".

Figure 11: Limiting the range of the **PercentageComplete** property

Besides validation, these extended properties can also be used to control how this property will be visualized in the client by default. The **Priority** property is stored as an integer, but visualized to the user in a more textual way by clicking on the **Choice List...** option in the **Properties** window.



The screenshot shows a configuration panel for "Name:". It has a text input field with the value "Priority". Below it is a link labeled "Choice List..." with a small "X" icon next to it.

Figure 12: Click the **Choice List...** link in the extended properties panel

This opens a pop-up where you can define the available choices and their display names.

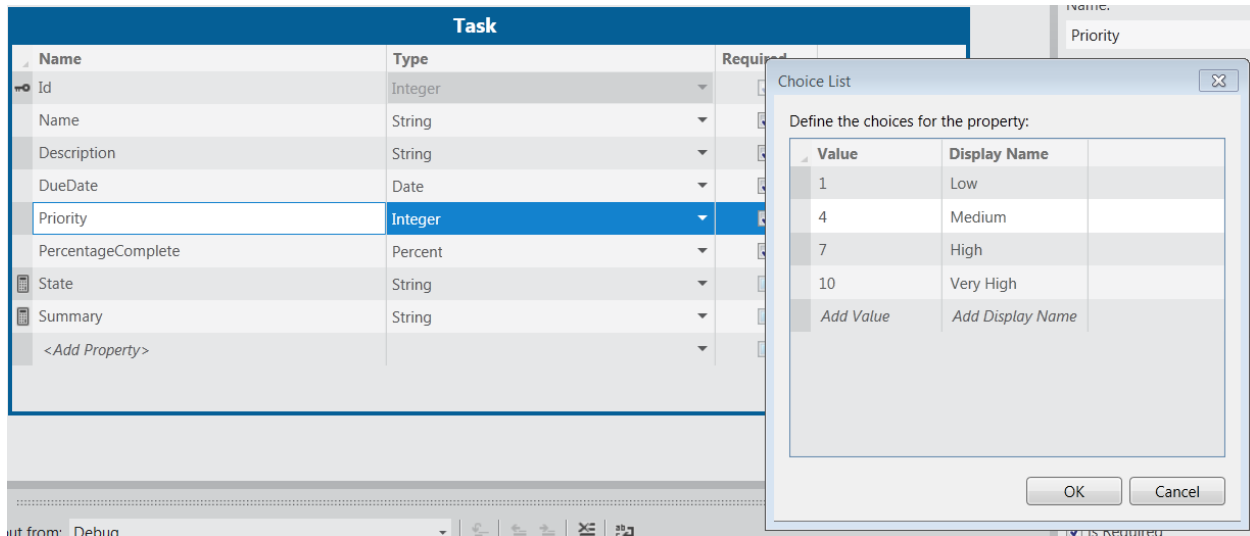


Figure 13: Limiting the allowed choices for the **Priority** property

You might notice that some properties have an icon next to their name. The icon that looks like a key is straightforward: it indicates a field that will be used as a primary key column in the database. The icon that looks like a calculator however, warrants some extra explanation.

LightSwitch supports computed properties. These are properties that are not stored in the database but instead calculated at run time on the tier (client or server) where they are accessed. Turning a simple property into a computed property is done by selecting it, and then selecting the **Is Computed** check box in the **Properties** window.

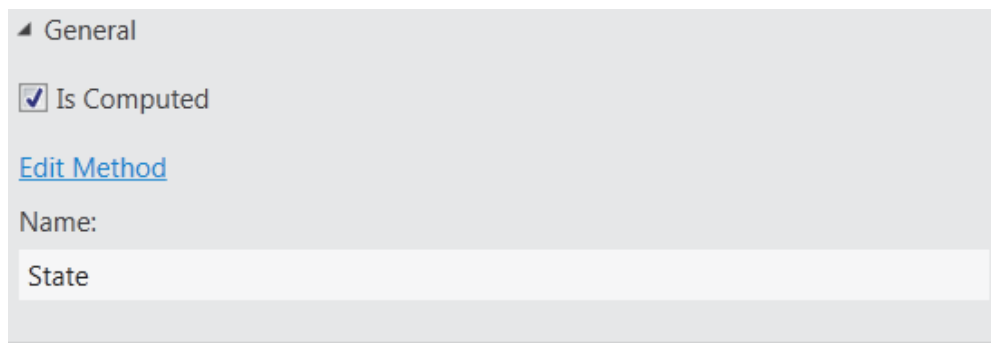


Figure 14: Making the **State** and **Summary** properties computed properties

Click on the **Edit Method** link to access a code editor where you can implement the business logic for how to calculate this property. In the following code sample, I fill the blank **State_Compute** method that LightSwitch created for me with the desired business logic.

```
namespace LightSwitchApplication
{
    public partial class Task
    {
        partial void State_Compute(ref string result)
        {
```

```

        if(this.PercentageComplete == 100){
            result = "Complete";
        }
        else
            if (this.DueDate > DateTime.Today) {
                if (this.PercentageComplete == 0){
                    result = "Pending...";
                }
                else {
                    result = "Started";
                }
            }
            else {
                result = "Overdue!";
            }
    }

    partial void Summary_Compute(ref string result)
    {
        result = this.Name + "(" + this.State + ")";
    }
}

```

The **State** property is a simple label based on the **PercentageComplete** and **DueDate** properties. For tasks, I decided to compute a **Summary** property as well to help represent the entity as a string. By default, if an entity needs to be represented as a single line, LightSwitch shows the value of the first **string** property. Overriding this convention can be done by selecting the entity itself and choosing the desired entity property for the **Summary Property** option in the **Properties** window.

The screenshot shows the Entity Designer interface with two tabs: 'General' and 'Appearance'. The 'General' tab is active, showing the following settings:

- Default Screen:** A dropdown menu set to 'Auto'.
- Plural Name:** A text box containing 'Tasks'.
- Is Searchable:** A checkbox that is checked.
- Name:** A text box containing 'Task'.

The 'Appearance' tab is also visible, showing the following settings:

- Description:** An empty text box.
- Display Name:** A text box containing 'Task'.
- Summary Property:** A dropdown menu set to 'Summary'. Below the dropdown is a list of properties: Id, Name, Description, DueDate, Priority, PercentageComplete, State, and Summary. The 'Summary' property is highlighted at the bottom of the list.

At the bottom of the window, there are three tabs: 'PROPERTIES' (which is selected), 'SOLUTION EXPLORER', and 'TEAM EXPLORER'.

Figure 15: For each entity, you can select the Summary property

At first glance, the Entity Designer looks like a database designer, but you should really try to think of it as a domain object designer, since the choices made here will have effects throughout the application, not just in the database tier. This becomes even more apparent in our third entity: *Person*.

Creating a Person entity with clever reuse of existing business types


Person			
Name	Type	Required	
 Id	Integer	<input checked="" type="checkbox"/>	
FirstName	String	<input checked="" type="checkbox"/>	
FamilyName	String	<input checked="" type="checkbox"/>	
NickName	String	<input type="checkbox"/>	
Phone	Phone Number	<input type="checkbox"/>	
Email	Email Address	<input type="checkbox"/>	
Avatar	Image	<input type="checkbox"/>	
 FullName	String	<input type="checkbox"/>	
<Add Property>		<input type="checkbox"/>	

Figure 16: Entity Designer for a Person Entity

In this entity, we'll stray away from the simple property types used before (**Boolean**, **Date**, **DateTime**, **Decimal**, **Double**, **Guid**, **Integer**, **Long Integer**, **Short Integer**, and **String**) and encounter some properties of types that you would not normally find in the normal database lingo (e.g., **Phone**, **Email**, and **Avatar**). In the LightSwitch Entity Designer, one property type is a **business type**. Besides the simple property types already listed, LightSwitch understands percentages, email addresses, phone numbers, the concept of money, what an image is, and how web addresses work. Business types come with specific validation rules, and often tailored controls are used by default to visualize this property.

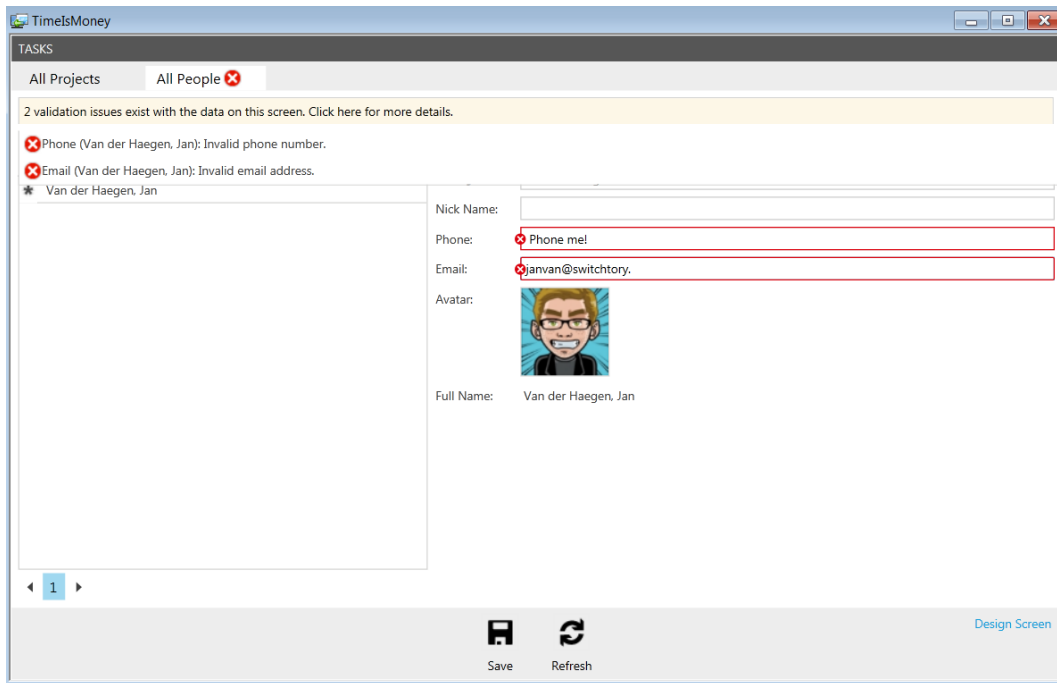


Figure 17: LightSwitch already understands some special business types

If there's a business type that your business needs but is not included in Visual Studio LightSwitch, you can easily create custom business types using the Extensibility Toolkit.

Taking full control

Also, remember that by using the **Write Code** button, you can take control of any part of the application. This extension point allows you to quickly add logic that controls how particular properties are computed as shown before, allows you to implement your business rules, custom validation, screen-level or application-level visualizations, and much more. I will later refer back to this as *level-one extensibility*.

One example of this extensibility we can add to our application is to add a fresh task each time a **Project** is created. The code for this is added by clicking on the **Write Code** button in the Entity Designer for the **Project** entity, and simply adding the following code:

```
namespace LightSwitchApplication
{
    public partial class Project
    {
        partial void Project_Created()
        {
            var firstTask = this.Tasks.AddNew();
            firstTask.Description = "A first task for every project!";
            //Fill in other required properties here.
        }
    }
}
```


This code will execute when end users create a new **Project**, no matter if they are using the client from the UI, the code behind, or even if they are connecting to the server directly with another client. We'll discuss that last scenario later.

Don't forget to design the relationships

Before finishing this brief introduction to the Entity Designer and exploring the application we've just built, we'll need to define the relationships between **Projects**, **Tasks**, and **People**. This is done by clicking on the **Add Relationship...** button in the Entity Designer toolbar.

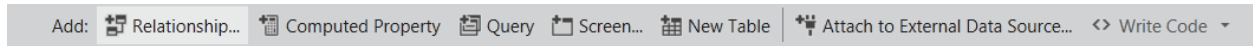


Figure 18: Using the Entity Designer toolbar to define relationships

In the **Add New Relationships** dialog that opens, you can define a new relationship between the selected entity and any other entity. We are now using the Entity Designer to model our own entities, for which tables will be generated. However, as will be described in a bit, you can also model LightSwitch entities over databases or other data sources that already exist. These might be legacy databases that do not have correct relationships defined and are not under your control to modify. By using the LightSwitch relationship designer, you can still design the relationships between these entities, or even between related entities from different data sources!

Create a new relationship between **Person** and **Task** such that one person can have many tasks, but a task can only be assigned to a single person. The following figure shows how this relationship can be defined.

Add New Relationship

Select two tables and properties to form a relationship between the two:

	From	To
Name:	Person	Task
Multiplicity:	One	Many
On Delete Behavior:	Restricted	
Navigation Property:	Tasks	AssignedTo

Person

Tasks

1

Task

Person

∞

A 'Person' **can have** many 'Task' instances.

A 'Task' **must have** a 'Person'.

'Person' **cannot be deleted**, if there are related 'Task' instances.

OK
Cancel

Figure 19: Defining a relationship between Task and Person

Similarly, let us create a new relationship between **Task** and **Project**. Because some tasks are grouped together and others are just small to-dos, the relationship between **Task** and **Project** is defined as a “zero-or-one to many” relationship.

Add New Relationship

Select two tables and properties to form a relationship between the two:

	From	To
Name:	Task	Project
Multiplicity:	Many	Zero or one
On Delete Behavior:		Restricted
Navigation Property:	Project	Tasks

Task

Project

Project

Tasks

∞

0..1

A 'Task' **can have** one 'Project'.

A 'Project' **can have** many 'Task' instances.

'Project' **cannot be deleted**, if there are related 'Task' instances.

OK
Cancel

Figure 20: Defining a relationship between Task and Project

While defining relationships, you will notice that some relationship types like “zero-or-one to zero-or-one”, “one to one” and “many to many” are not directly supported in LightSwitch, but they can often be accomplished through the use of a link table and by making use of level-one extensibility.

To play around with the application we have designed so far (I intentionally use the term “designed” and not “coded”), add two **List and Details** screens called **AllPeople** and **AllTasks** for the respective entities by following the instructions on creating screens in [Chapter 2](#).

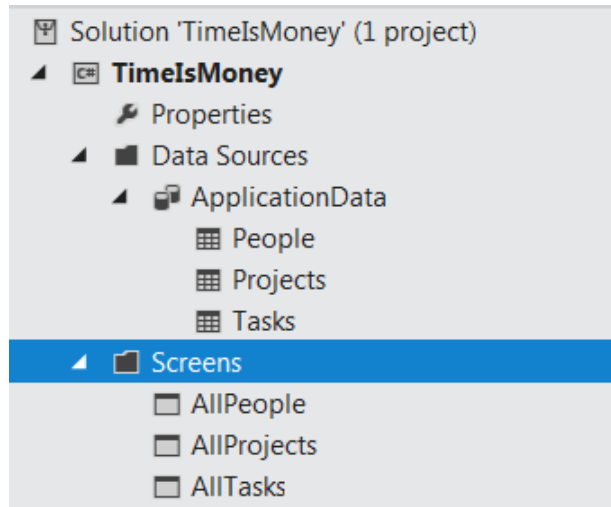


Figure 21: Application with three entities and corresponding screens

Congratulations, you now have a fully working application to help you take back control of your time! As you can see in the following figure, I have added two tasks to my application: *Create some entities* and *Do the dishes*. Feel free to create your own test data and tasks.

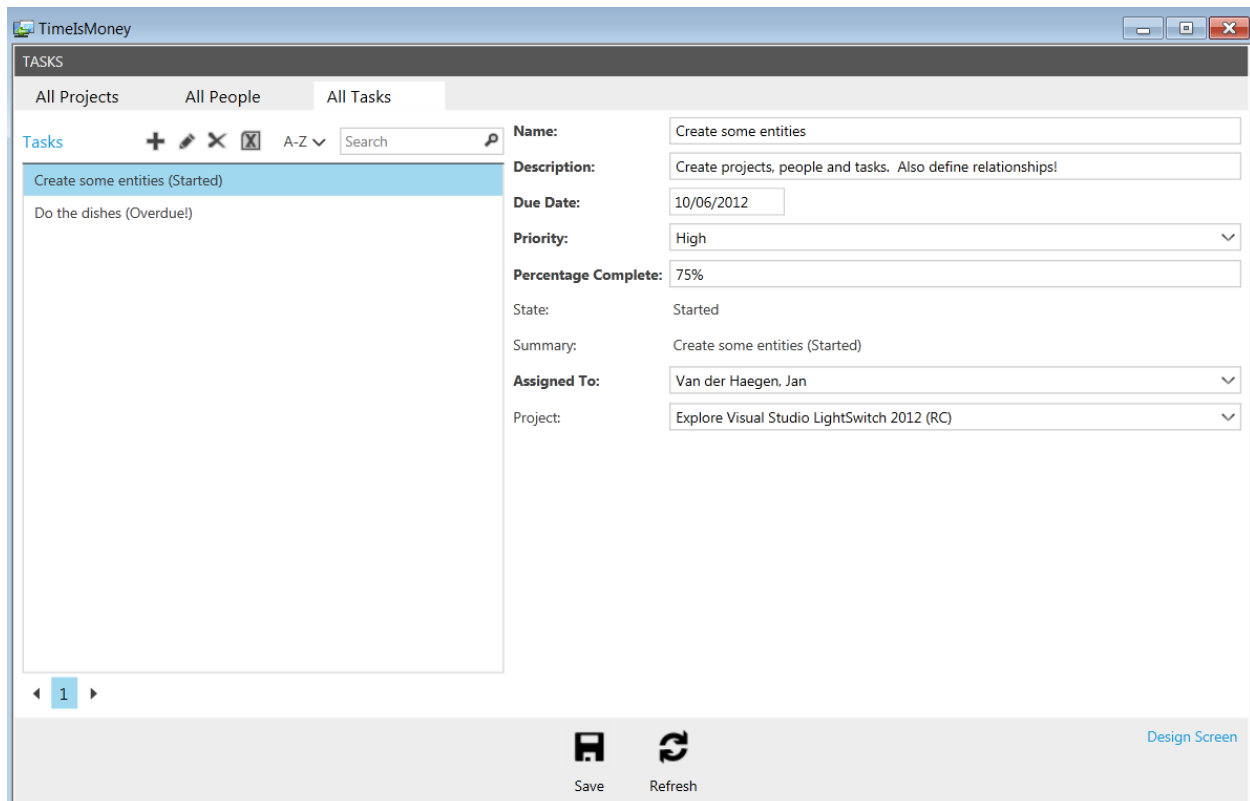


Figure 22: Our first application

Before moving on, I want to emphasize that the Entity Designer should be considered a business entity (domain model) designer, not a database designer.

The business entities designed in the Entity Designer, their properties, the business types of their properties, and the extended properties thereof, including the validation, are used by LightSwitch throughout the application in the data tier, on the server, and on the client.

Data-in

It's a common scenario that the data from one application has to be accessible in another application. LightSwitch applications are far from isolated data silos, and the IDE has out-of-the-box support for both data-in and data-out scenarios.

To explore the former, remember that we conveniently clicked on the **Create New Table** link on the LightSwitch home screen and started designing our first entity.

These entities will be stored in an intrinsic data source called **ApplicationData** by default.

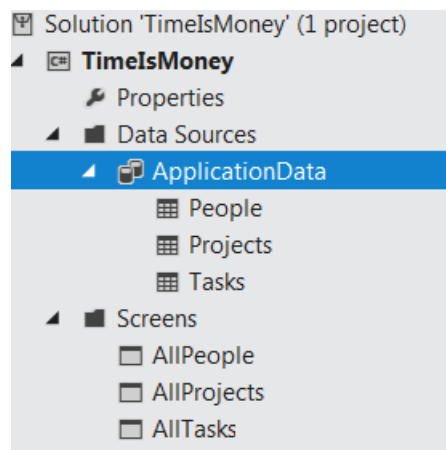


Figure 23: There can be many data sources in a LightSwitch project

If you want to use LightSwitch to work with existing data, use the second link on the LightSwitch home screen—**Attach to an external Data Source**. This will start a wizard to help you connect to one of the following sources:

- An existing database.
- A SharePoint site.
- An OData endpoint.
- A WCF RIA service.

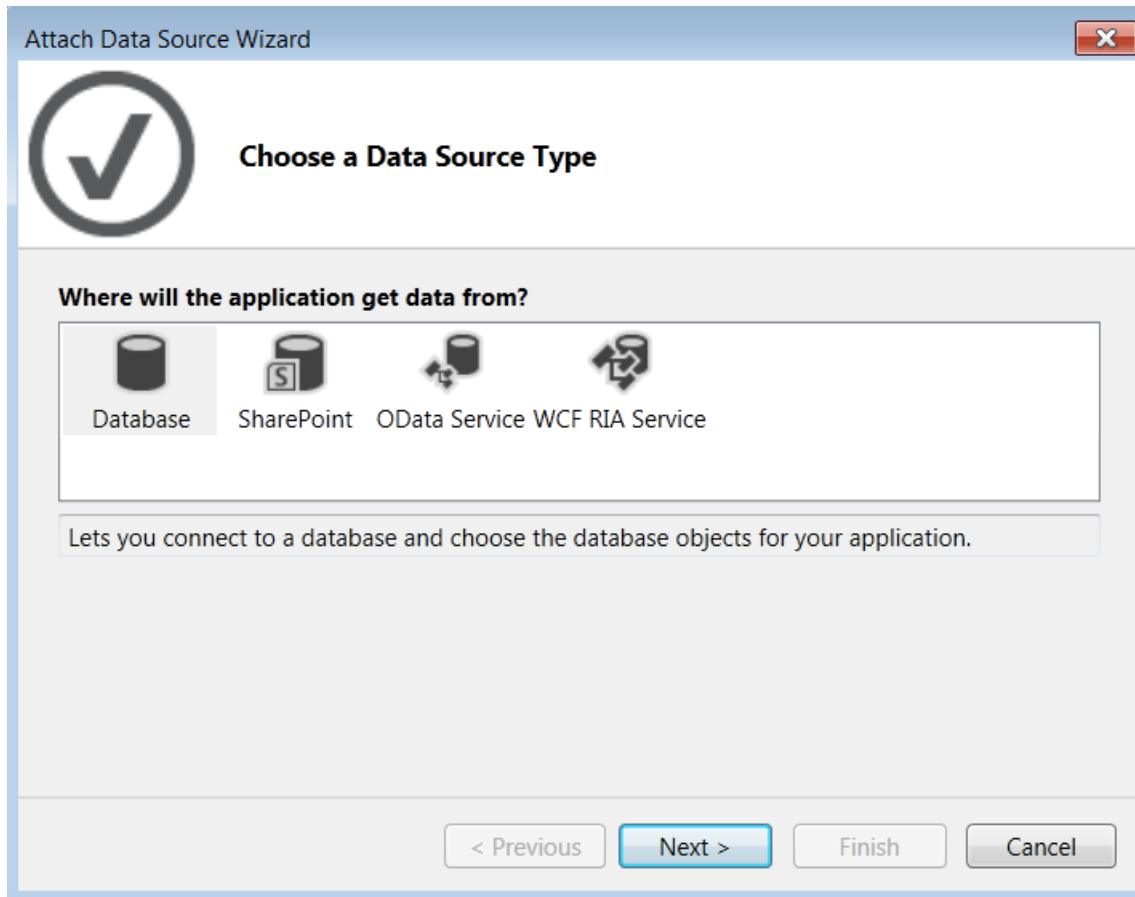


Figure 24: The Data Source wizard helps you connect to external data sources

The **Attach Data Source Wizard** will add a non-intrinsic data source to your application and allow you to design your business entities over the data with some validation limitations (options like “is the property required”, “maximum length”, and the compatible Business Type of the properties are limited depending on the source data type).

To OData or not to OData

The option to connect to an OData service in LightSwitch 1.0 has been kept for Visual Studio LightSwitch 2012.



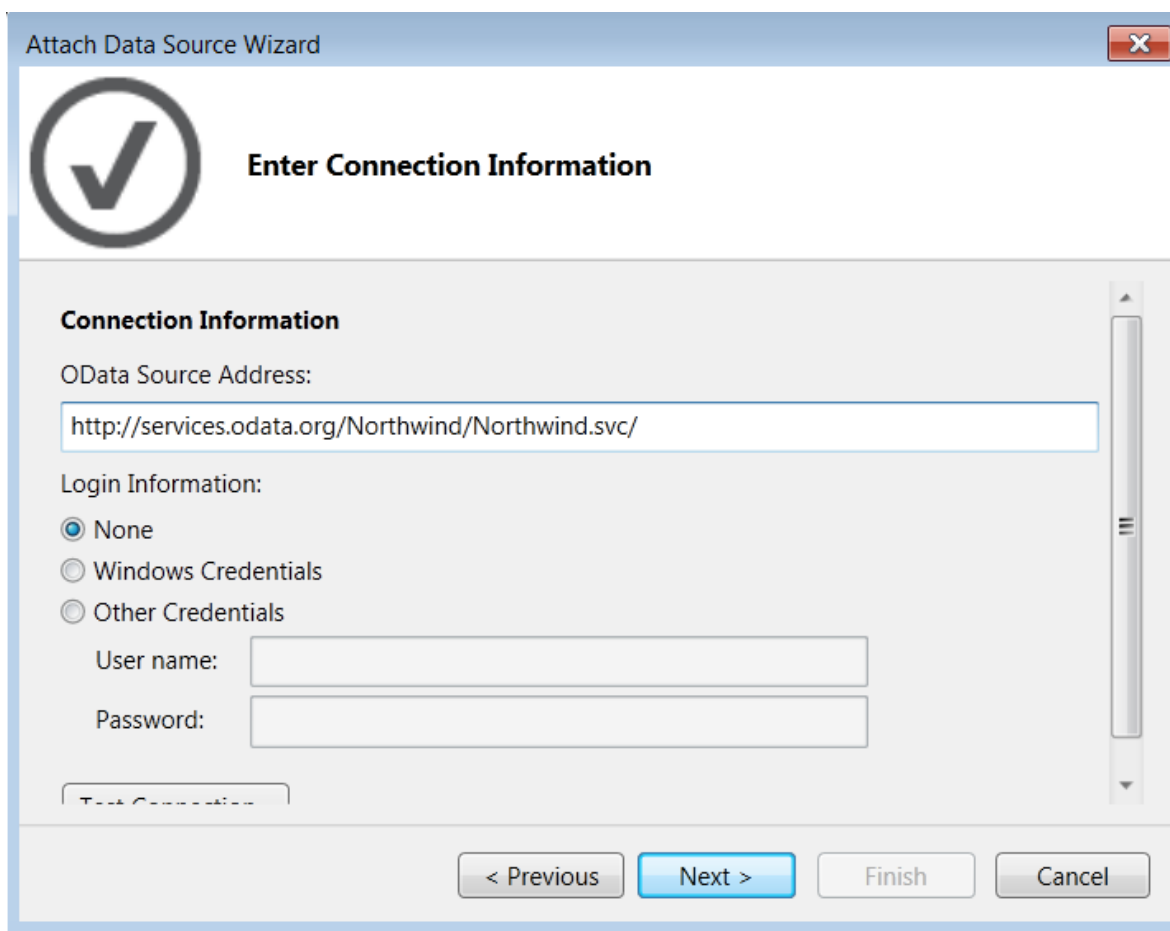
Figure 25: Open Data Protocol

OData, short for Open Data Protocol, is an industry standard that has been emerging fast over the last couple of years and is now widely adopted as a standard way to exchange data over the web. More technically, it's a protocol defined by Microsoft that defines a REST implementation. On the OData webpage you can find an ever-growing [list of OData producers](#), including SSRS

(SQL Server Reporting Services), Microsoft Dynamics CRM 2011, Windows Azure table storage, and the famous Northwind database.

If you want to test this data-in scenario, select the **OData Service** option in the **Attach Data Source Wizard** and then click **Next**.

In the **Connection Information**, use the following OData endpoint as your **OData Source Address**: <http://services.odata.org/Northwind/Northwind.svc/>. This endpoint requires no authentication, so select **None** as the selected **Login Information**, and then click **Next**.



The screenshot shows the 'Attach Data Source Wizard' window with the title bar 'Attach Data Source Wizard' and a close button. The main area has a large checkmark icon and the heading 'Enter Connection Information'. Below this, the 'Connection Information' section contains an 'OData Source Address' field with the text 'http://services.odata.org/Northwind/Northwind.svc/'. The 'Login Information' section has three radio buttons: 'None' (selected), 'Windows Credentials', and 'Other Credentials'. Below these are 'User name' and 'Password' text boxes. At the bottom of the wizard area is a 'Test Connection' button. The bottom of the window features four buttons: '< Previous', 'Next >' (highlighted in blue), 'Finish', and 'Cancel'.

Figure 26: Connecting to an external OData service

Once a successful connection has been made, LightSwitch investigates the metadata of your data source and allows you to import one or more data entities as business entities.

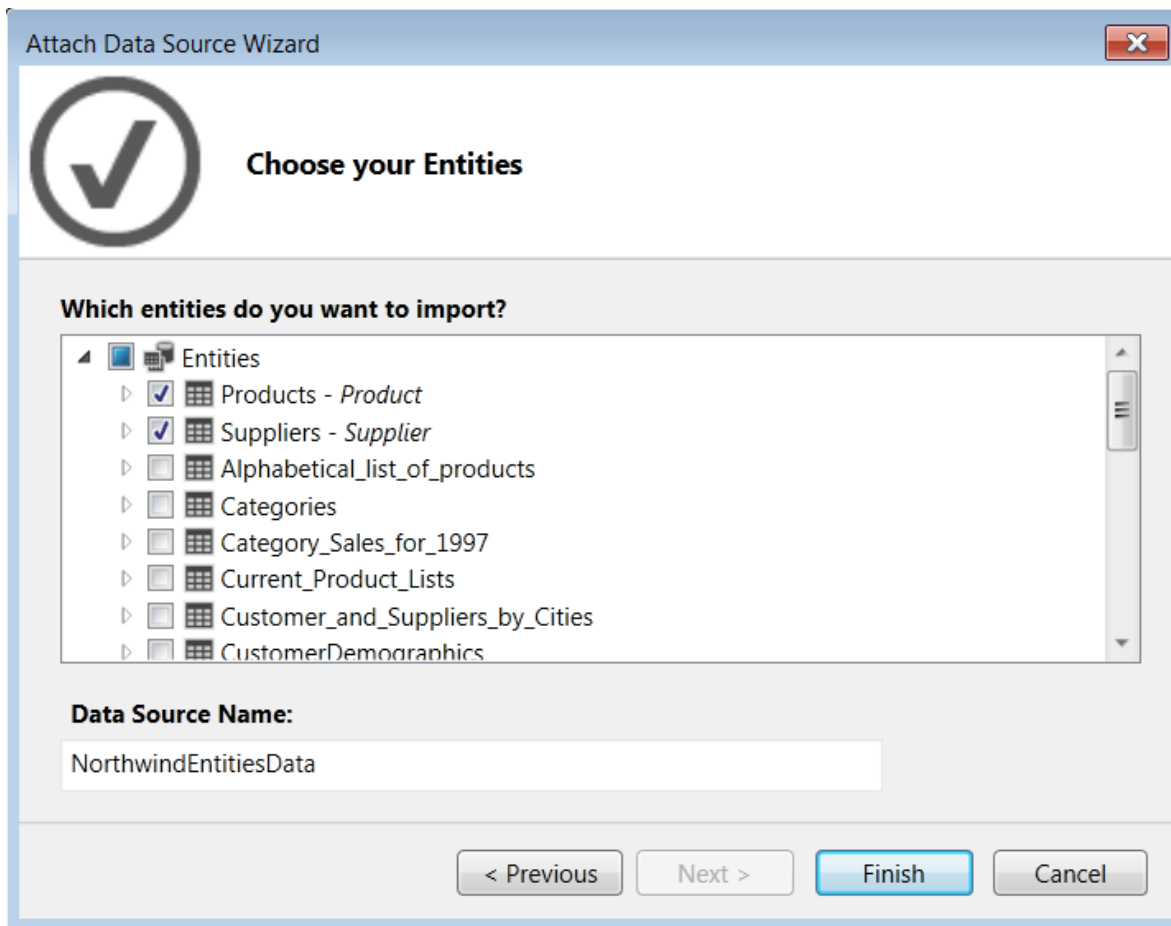


Figure 27: Defining which entities to import

Select the **Suppliers** check box; **Products** will also be imported automatically because of the relationship between the two, and then click **Next** to finish the wizard.

Notice that although the data type of the **Supplier** entity's **Phone** property is **String**, you can use the **Entity Designer** to remap it to the **Phone Number** business type.

Supplier			
Name	Type	Required	
SupplierID	Integer	<input checked="" type="checkbox"/>	
CompanyName	String	<input checked="" type="checkbox"/>	
ContactName	String	<input type="checkbox"/>	
ContactTitle	String	<input type="checkbox"/>	
Address	String	<input type="checkbox"/>	
City	String	<input type="checkbox"/>	
Region	String	<input type="checkbox"/>	
PostalCode	String	<input type="checkbox"/>	
Country	String	<input type="checkbox"/>	
Phone	String	<input type="checkbox"/>	
Fax	Email Address	<input type="checkbox"/>	
HomePage	Phone Number	<input type="checkbox"/>	
Products	String	<input type="checkbox"/>	
<Add Property>	Web Address	<input type="checkbox"/>	

Figure 28: Remapping data types to business types

Data-out

Another out-of-the-box feature is the supported data-out scenario. For each of your data sources, intrinsic or not, LightSwitch will generate and host an OData endpoint that exposes your business entities.

The endpoint can be found at `http://<UriToTheApplication>/<DataSourceName>.svc`.

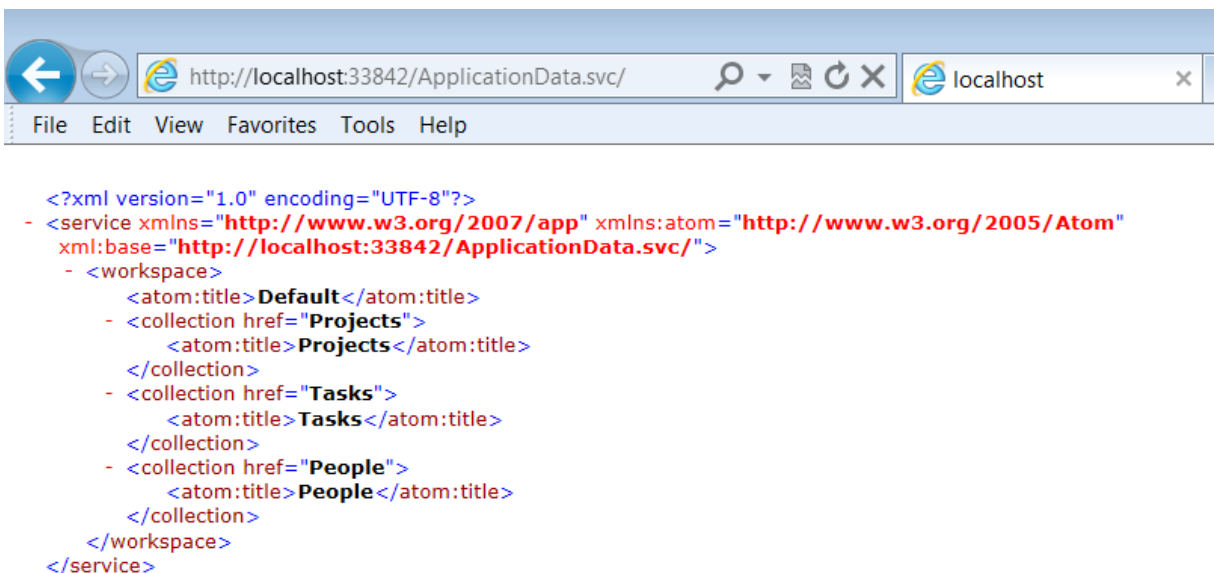


Figure 29: Browsing to the OData endpoint

If the generated Silverlight application doesn't fit your business needs, you can reach these OData endpoints in numerous other client technologies, including (but not limited to):

- Excel PowerPivot: [Creating and Consuming LightSwitch OData Services](#)
- Windows 8: [Using LightSwitch OData Services in a Windows 8 Metro Style Application](#)
- Windows Phone: [Consume a LightSwitch OData Service from a Windows Phone Application](#)
- HTML/JavaScript: [A Full CRUD Data.js and Knockout.js LightSwitch Example Using Only an HTML Page](#)
- jQuery Mobile: [A Full CRUD LightSwitch jQuery Mobile Application](#)
- Android: [Communicating with LightSwitch Using Android App Inventor](#)
- Unity 3D: [Using Visual Studio LightSwitch to Orchestrate a Unity 3D Game](#)

One important thing to note about these OData endpoints is that they do not simply expose your data, but expose your business entities with respect to their business logic, security settings, and default or coded validation rules.

Blending data-in and data-out

One powerful way to use LightSwitch is to use it in combination with other technologies, combining both data-in and data-out scenarios. Because of the choice to use OData in the LightSwitch middle-tier, LightSwitch can be used not only to create simple CRUD applications, but to accomplish complex data ecosystems.

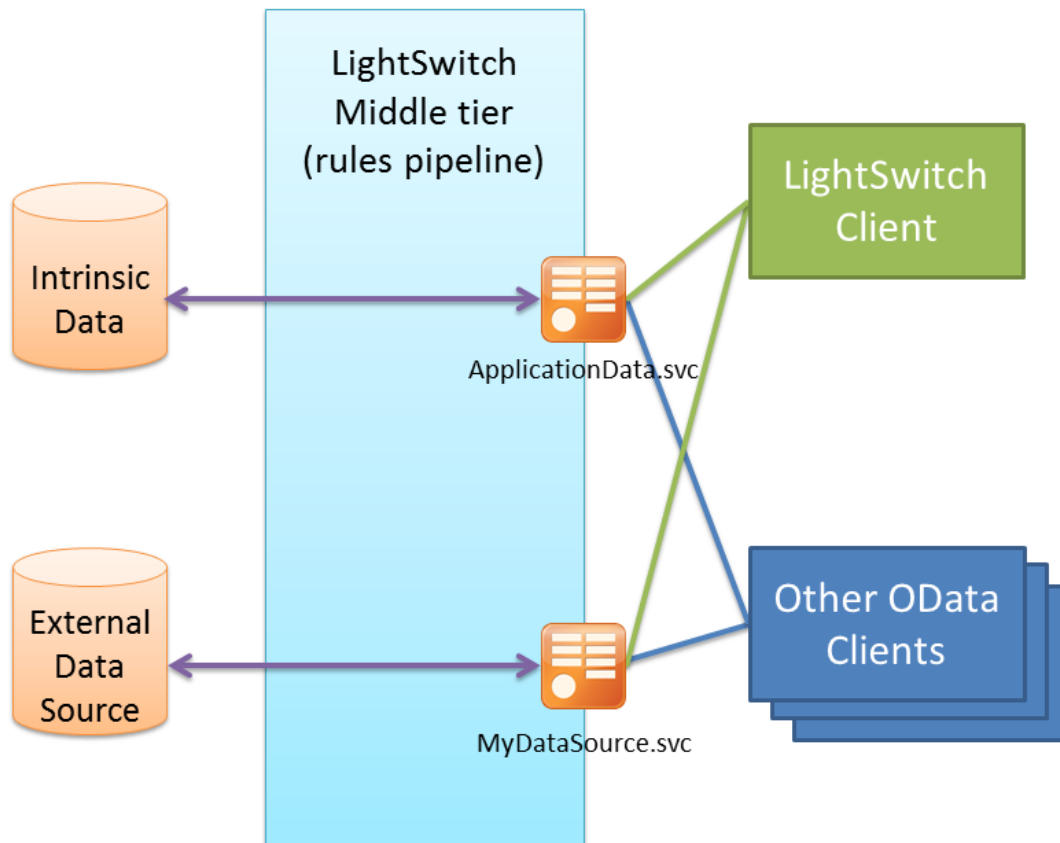


Figure 30: LightSwitch mash-ups between data-in and data-out scenarios

Another powerful way to use LightSwitch is to use it solely for its data-in and data-out functionality. This makes it easy to convert an older WCF RIA service or SQL database to an OData service, or offer a limited view on your internal SharePoint site to an external party.

Both the image above and the links were gathered by Beth Massi (see [Chapter 7](#)).

Chapter 4 The Query Editor

Finding the tasks that matter

Because our three screens show all available entities, it's easy to imagine that the screens in their current state will become useless pretty fast as they are filled with hundreds of old and already completed tasks. Most of the time, we will not be interested in all tasks, but only in the ones that have to be completed next.

For any entity, LightSwitch automatically generates a **Select all** and a **Select by ID** query, but to filter out the old and unimportant tasks successfully, one has to create a custom query. Like most things in LightSwitch, this will only take a couple of seconds once you get used to the **Query Editor**.

In the **Solution Editor**, right-click on the **Tasks** entity and select **Add Query**.

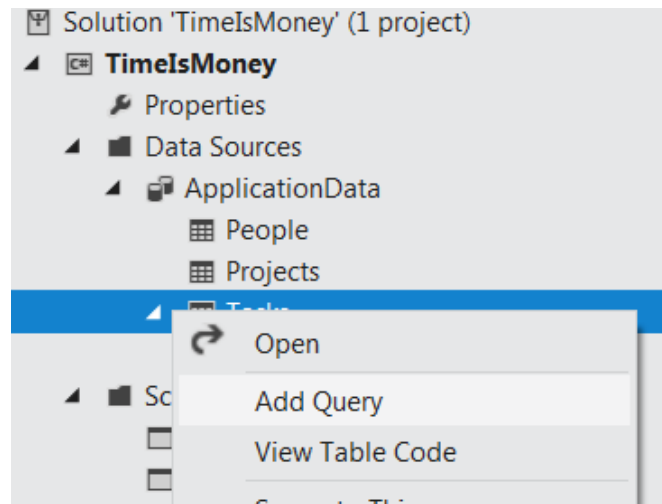


Figure 31: Adding a query for an entity from the **Solution Explorer**

The first thing you'll want to do is to rename the query as something more meaningful, **UrgentTasks** for example. This can be done in the **Properties Window**.

General

Number of Results Returned:
Many

Name:
UrgentTasks

[Edit Additional Query Code](#)

Source:
Tasks

Return Type:
Task

Appearance

Description:

Display Name:
Urgent Tasks

Figure 32: A query has extended properties too

Adding filter criteria

The first thing we'll want to do is to add a new **Single** filter.

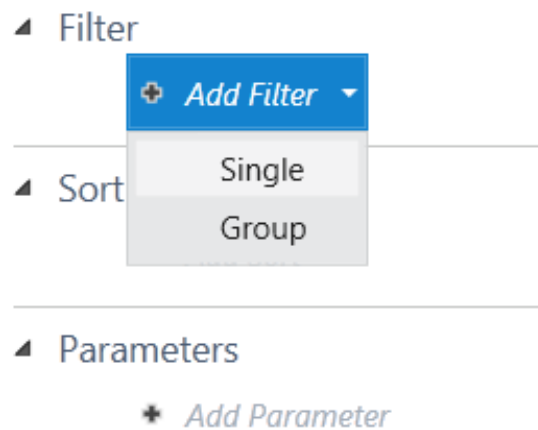


Figure 33: Add a new **Single** filter

In this query, we do not want to see any tasks that are in the distant future. The first filter criteria should be that the **DueDate** is less than the end of the month.

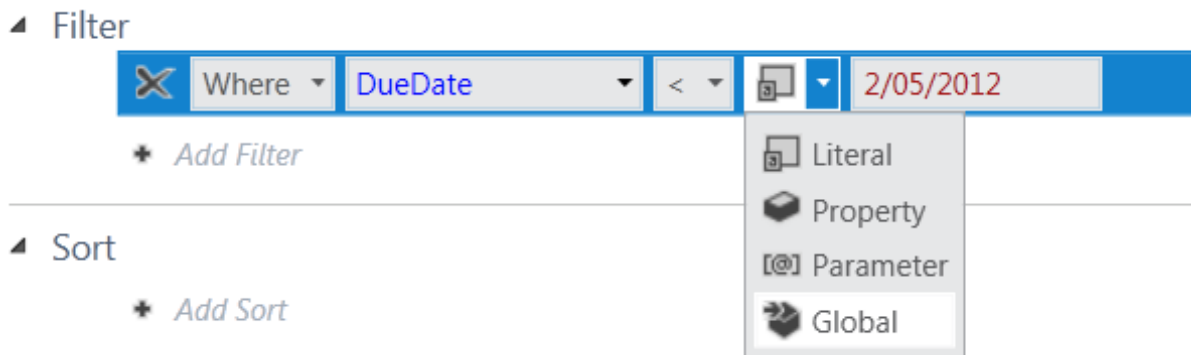


Figure 34: Filtering based on DueDate

We'll also want to filter out any tasks that have been completed and any tasks with a priority less than 4. You might remember that the business type of the **Priority** property is an integer even though we represent it to the user as a choice list with "Low," "Medium," "High," and "Very High" labels.

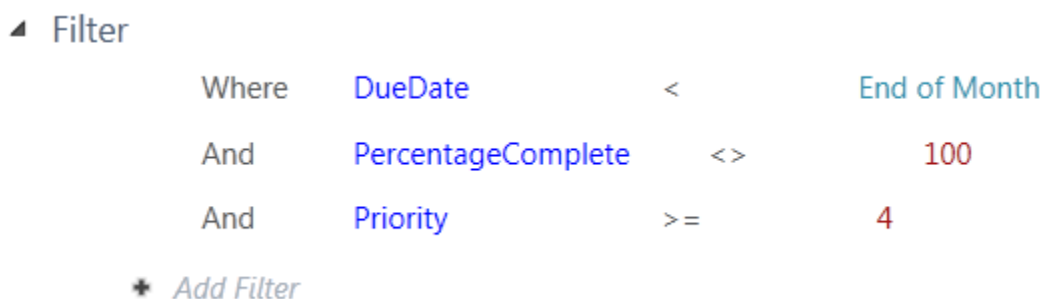


Figure 35: Filtering based on DueDate, PercentageComplete, and Priority

Adding some logical sorting

While we're in the **Query Designer**, finish the query by adding a default sort for our urgent tasks based on **Priority**, and then by **DueDate**.

Filter

Where	DueDate	<	End of Month
And	PercentageComplete	<>	100
And	Priority	>=	4

* Add Filter

Sort

Sort by	Priority	Descending
Then by	DueDate	Ascending

* Add Sort

Figure 36: Adding a default sort to the query

Add a screen directly to this query

To test your query, you can create a fourth **List and Details** screen called **AllUrgentTasks**. Because the query is the default global query, you can select it as the **Screen Data** to use. (LightSwitch also allows queries to be defined at screen level, called local queries.)

Add New Screen

Select a screen template:

- Details Screen
- Editable Grid Screen
- List and Details Screen
- New Data Screen
- Search Data Screen

List and Details Screen
Displays a list of items and details for the selected list item

Provide screen information:

Screen Name: AllUrgentTasks

Screen Data: - UrgentTasks

(None)
People
Projects
Tasks
- UrgentTasks

OK Cancel

Figure 37: Adding a screen directly on a global query

With my test data, it seems to work correctly. Out of my two tasks, *Do the dishes* has a low priority, so it is not included in the **All Urgent Tasks** screen.

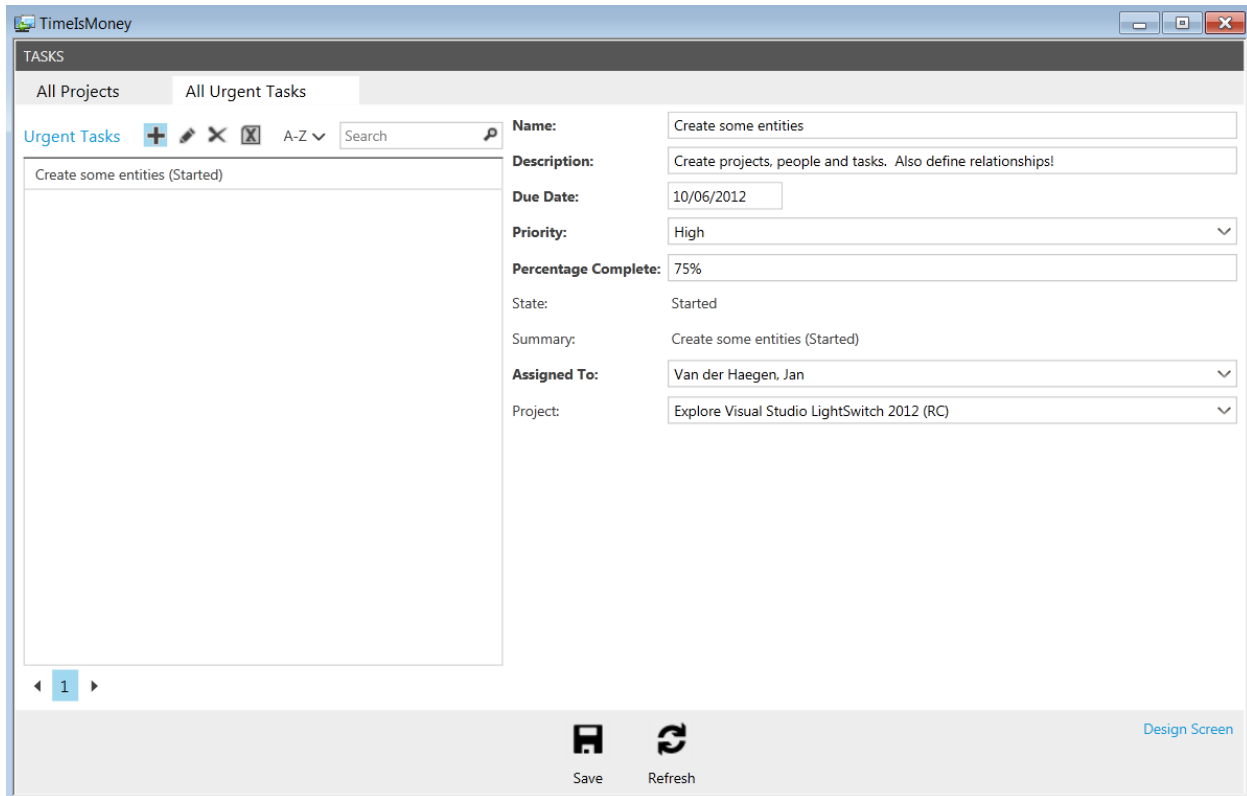


Figure 38: The new screen only shows the urgent tasks

Finding the important tasks per user

The query in Figure 38, however, displays all urgent tasks. This screen is great for me, as a manager of my own personal life, and the only user of this application so far. But if you were to use this application to manage several people, tasks, and projects simultaneously, you may be interested in an urgent-tasks-per-user query rather than the screen in Figure 38.

Create a second query by right-clicking your **Tasks** in the **Solution Explorer**, and selecting **Add New Query**.

Rename the query as **MyUrgentTasks**.

Query inheritance

Note that this query uses the complete **Tasks** list as the source, but we already have a query to separate the important tasks from the trivial ones. As an avid fan of avoiding repetition, I am thrilled that LightSwitch offers the possibility to base one query on the results of another. To do so, change the **Source** from **Tasks** to **UrgentTasks** at the top of the **Query Editor**.

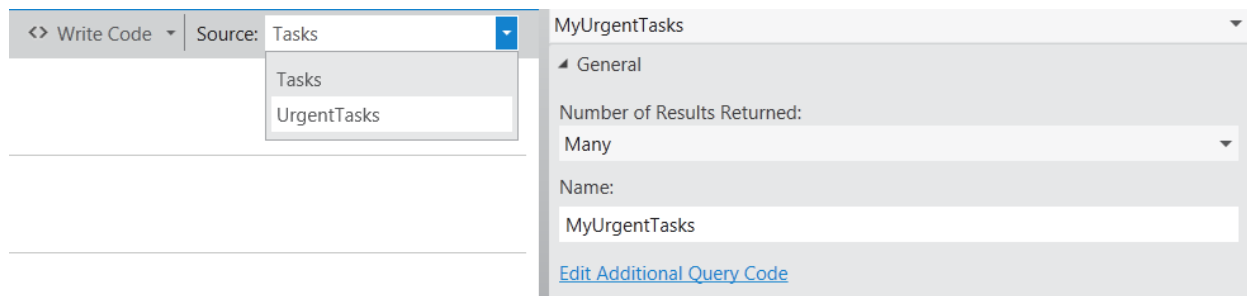


Figure 39: Inheriting a query from a previously made query

By doing so, this query will inherit the **UrgentTasks**' filtering and sorting criteria, allowing us to not worry about sorting or filtering out only the important tasks anymore. Our only concern then is finding the tasks assigned to a particular **Person**.

Adding parameters to a query

To do so, we'll add a new filter.

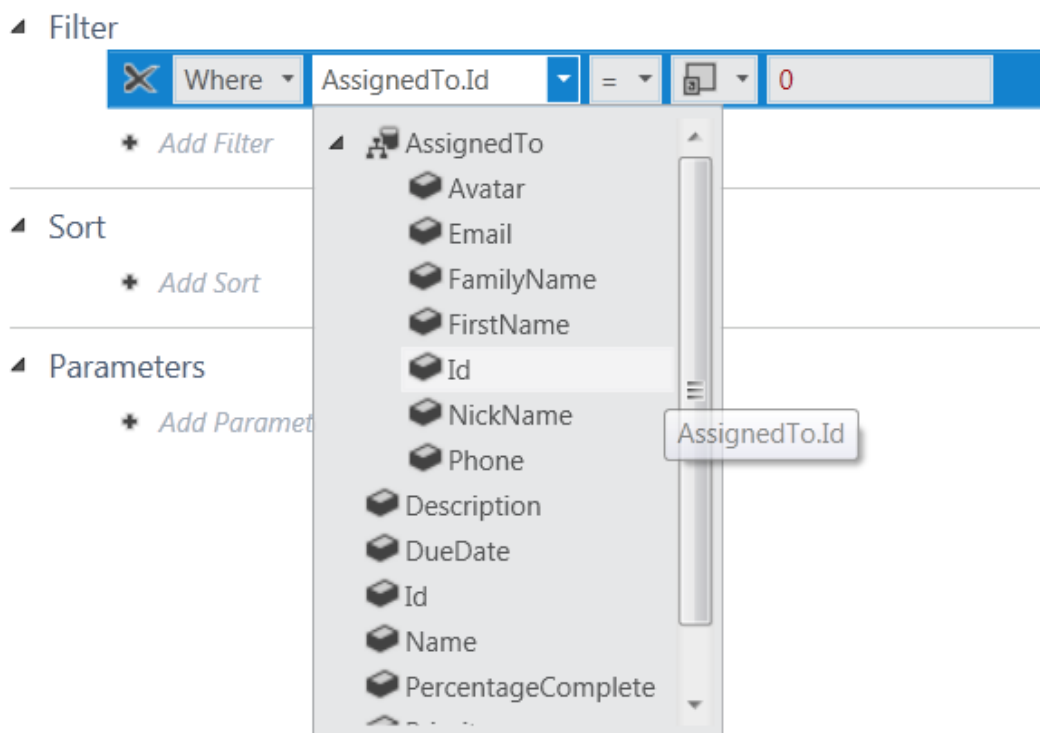


Figure 40: Filtering based on the person to which the task is assigned

Then select **Parameter** from the drop-down as shown in Figure 40, and add a new query parameter as shown in Figure 41.



Figure 41: Picking a source for the assignment



Figure 42: Setting the source to a new parameter

Rename the parameter as **AssignedPerson**.

To allow even more detailed control over which **Tasks** will be included, I added two additional filters and matching parameters: **MinimumPercentageComplete** and **MaximumPercentageComplete**. For these two however, I checked the **Is Optional** check box in the **Properties** window.

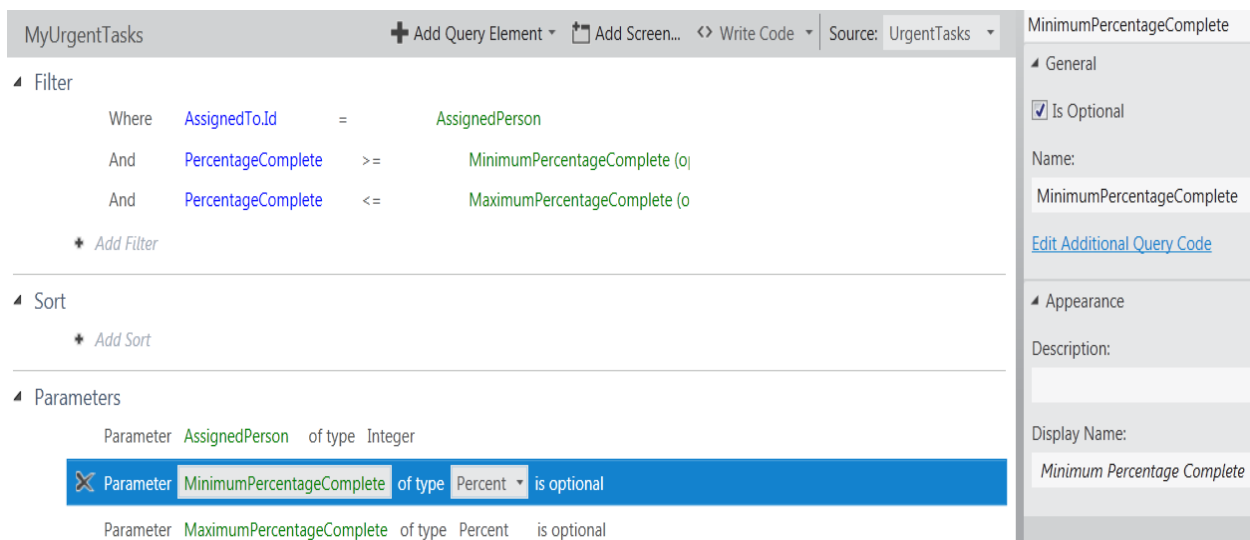


Figure 43: An overview of the query

To view this data, we'll have to add a new screen. This time the default screen generated by the **List and Details Screen Template**, which we've used three times now, won't be sufficient. It's time to have a closer look at the **Screen Editor**.

Chapter 5 The Screen Editor

Creating a Search Data Screen for our query

In the **Add New Screen** wizard, select the **Search Data Screen** template. Name the screen **MyUrgentTasks**, and select **MyUrgentTasks** as the **Screen Data**.

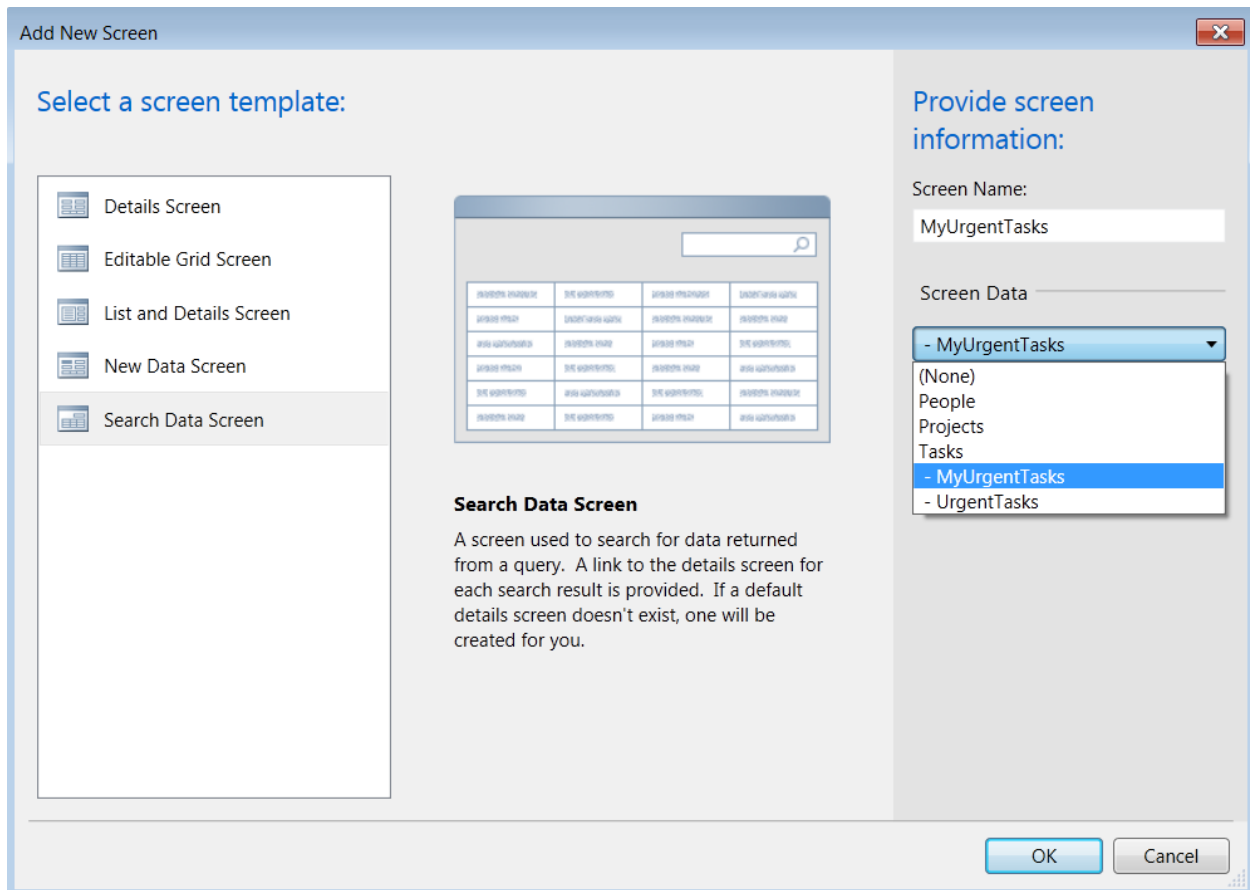


Figure 44: Creating a Search Data Screen

The Model and ViewModel part of the Screen Editor

In the left-hand side of the **Screen Editor**, LightSwitch added the two query parameters, but it added each parameter twice. The left side is considered the ViewModel and Model layer of the screen in the LightSwitch MVVM implementation. The first time you recognize these parameters, they actually represent the query parameters and are thus part of the Model, and the second time they are listed represents the screen properties as part of the ViewModel.

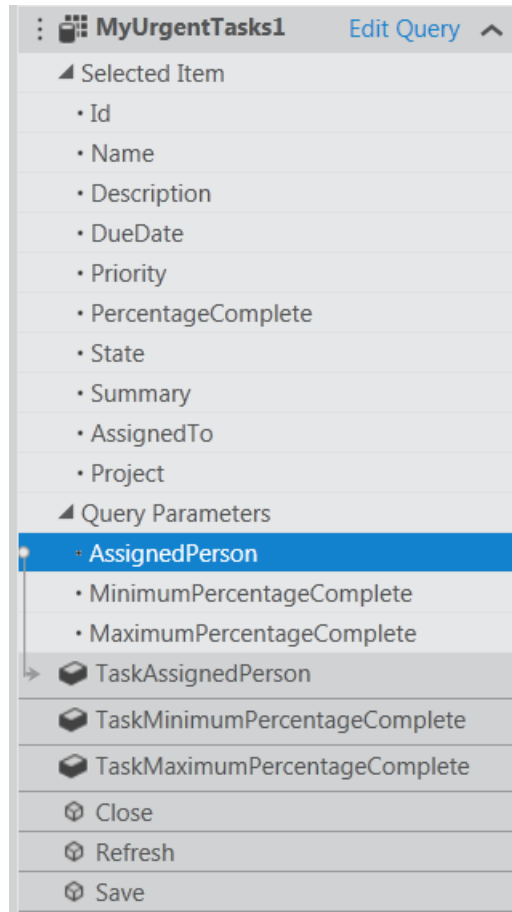


Figure 45: Left-hand side of the **Screen Designer** shows the **ViewModel** and **Model** layer

As shown in Figure 45, an arrow is displayed between the corresponding query parameters and the local screen properties to reveal that they are data-bound. This can also be observed in the **Properties** window by selecting one of the query parameters.



Figure 46: Binding properties

Turning a Screen Property into a required Screen Parameter

Select the **TaskAssignedPerson** screen property and in the **Properties** window, make sure both the **Is Parameter** and **Is Required** check boxes are selected.

☒ Is Parameter

▲ Formatting

Format Pattern

[About format patterns](#)

▲ Validation

Maximum Value

Minimum Value

[Custom Validation](#)

☒ Is Required

Figure 47: Turning a screen property into a required screen parameter

Also verify that both check boxes are cleared for the **TaskMinimumPercentageComplete** and **TaskMaximumPercentageComplete** local screen properties.

Validation on Screen Properties

The **Screen Properties** window allows you to set up some validation logic too. Change the number of **Percent Decimal Places** to **0**, and set the **Minimum Value** and **Maximum Value** to **0** and **100** respectively for the two percentage complete properties.

☐ Is Parameter

▲ Appearance

Percent Decimal Places

0

▲ Validation

Maximum Value

100

Minimum Value

0

[Custom Validation](#)

☐ Is Required

Figure 48: Screen properties can have validation too

You'll notice that there are some similarities between how these local screen properties work and how the **Entity Designer** works. With the **TaskMaximumPercentageComplete** local screen property still selected, click on the **Custom Validation** link in the **Properties** window.

We'll use this to write some simple code that checks if the entered minimum value is less than the entered maximum value.

```
public partial class MyUrgentTasks
{
    partial void
TaskMaximumPercentageComplete_Validate(ScreenValidationResultsBuilder
results)
    {
        if (this.TaskMinimumPercentageComplete >
this.TaskMaximumPercentageComplete) {
            results.AddPropertyError("Minimum cannot exceed the
maximum");
        }
    }
}
```

Initializing Screen Properties

By the way, while we're writing code, it might be a good idea to initialize our local screen properties. To do so, locate the **Write Code** button at the top of the **Screen Designer**, and click on the drop-down arrow. Click on the link **MyUrgentTasks_Created**—this will take you to the code extensibility point that runs on the client once the screen is created.

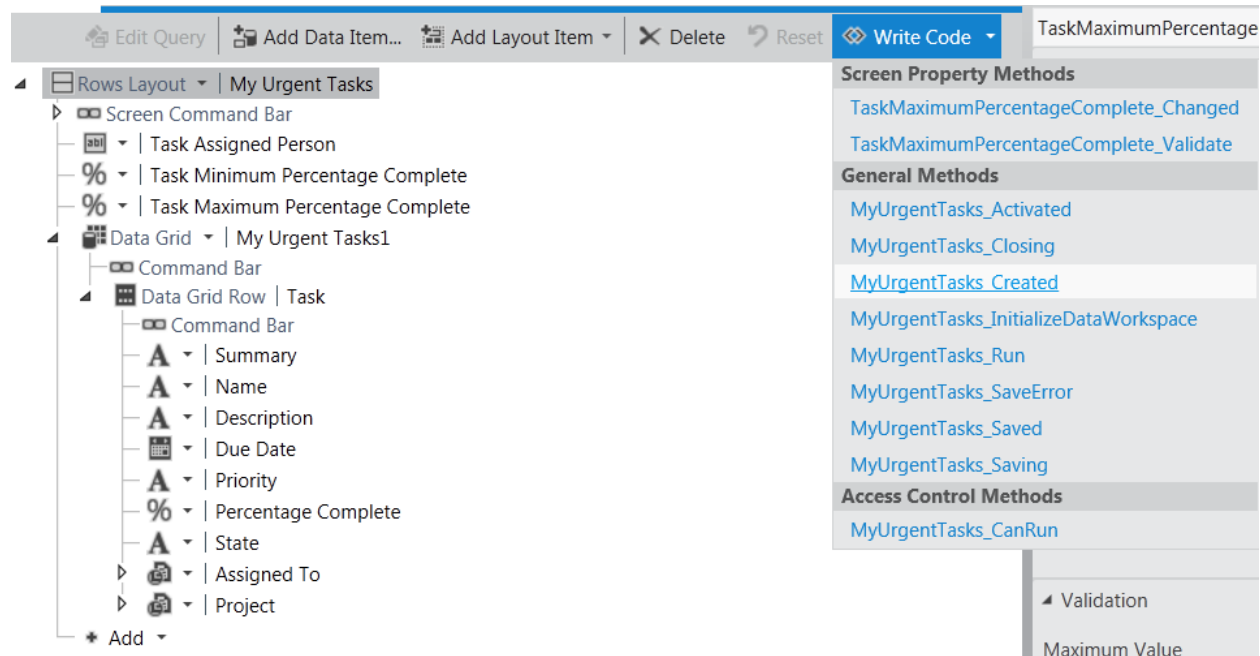


Figure 49: Finding the level-one extensibility point to initialize the screen properties

The code to initialize the two local screen properties isn't very surprising:

```
partial void MyUrgentTasks_Created()
{
    this.TaskMinimumPercentageComplete = 0;
    this.TaskMaximumPercentageComplete = 1;
}
```

Where did the screen go?

If you press F5 to debug your application, you'll notice that the navigation menu doesn't include our new screen.

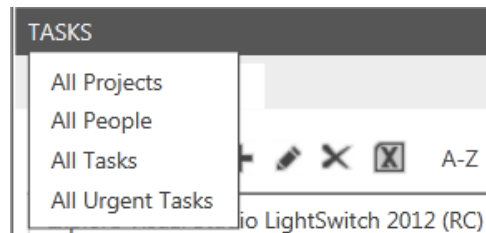


Figure 50: The new screen doesn't seem to be included in the navigation menu

The reason for this is that our new screen has one local screen property, **TaskAssignedPerson**, for which we selected both the **Is Required** and the **Is Parameter** extended properties. This means the code that opens this screen must pass an integer parameter (**TaskAssignedPerson**), and the navigation menu has no idea how to resolve a value for it.

We'll have to add our new screen manually, which will only be the fifth time we've had to code in this tutorial—not bad for how much functionality we have already built in this application.

Adding a new command to a screen

One suitable place to open this screen with urgent tasks from a particular person is the **All People** screen. Open the **All People** screen in the **Screen Editor** and locate the **Screen Command Bar** element near the top of the **Screen Visual Tree**, the center part of the **Screen Editor**.

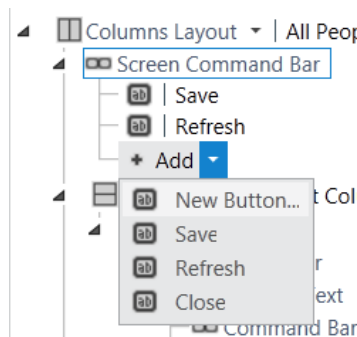


Figure 51: Adding a new button to the screen

Choose the **New Button...** option. In the pop-up that appears, select the **New Method** option and name it **ShowUrgentTasks**.

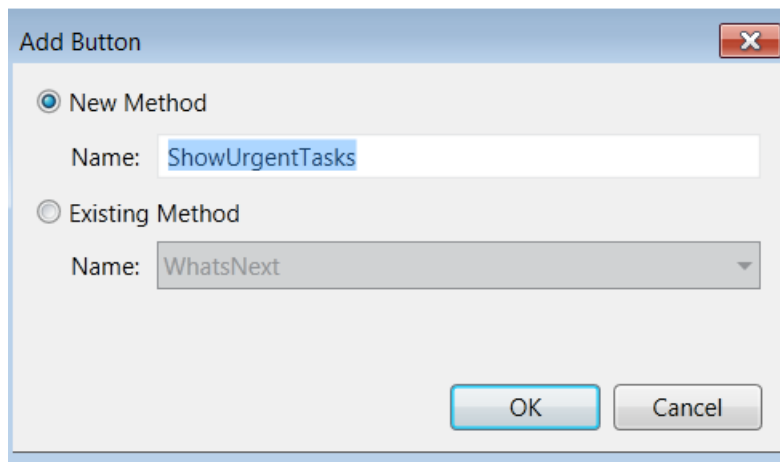


Figure 52: Creating a new command on the screen

A new command will be added, and like almost any element of your LightSwitch application, you can use the designer to set several extended properties. One property we'll definitely want to set is the image that is used. Personally, I've really become a fan of Syncfusion's free custom icon tool, Metro Studio (<http://www.syncfusion.com/downloads/metrostudio>). I use the icons almost exclusively for LightSwitch commands because of the great fit they are with the LightSwitch Cosmopolitan theme (see [Chapter 6](#)).

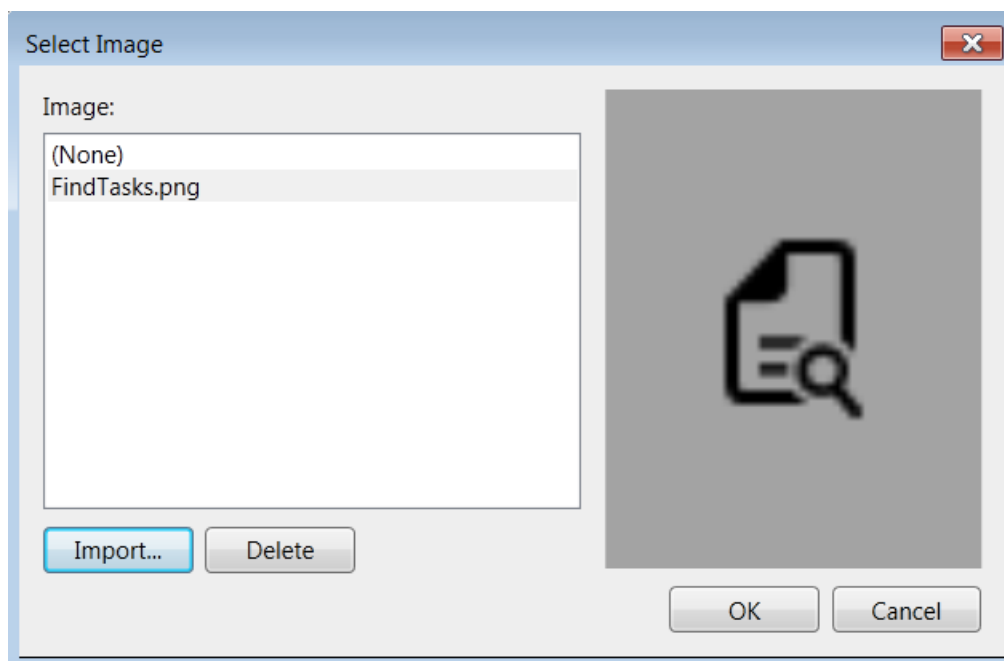


Figure 53: Adding an image to the command

After this is done, right-click on your button in the **Visual Tree editor**, and select the **Edit Execute Code** option from the context menu.

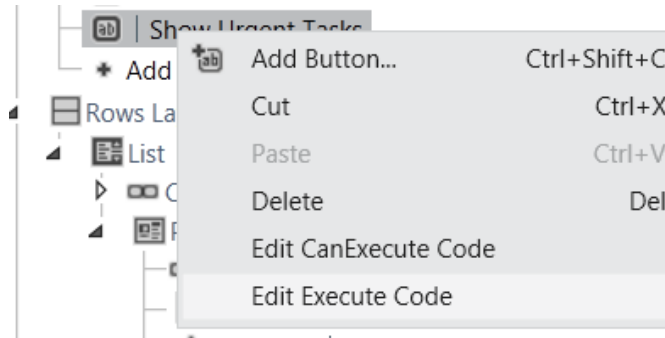


Figure 54: Editing the code behind the command

Opening a screen from code

LightSwitch generates several classes that you can use to code against if you choose to do so. For example, LightSwitch generates a method called **Show<ScreenName>** on the **Application** class for each screen. In our case, this makes the code that opens the screen as simple as:

```
public partial class AllPeople
{
    partial void ShowUrgentTasks_Execute()
    {
        Application.ShowMyUrgentTasks(this.People.SelectedItem.Id);
    }
}
```

This time, press F5 to pick the fruit of our labor. The **All People** screen now has an extra command as shown in the following figure.

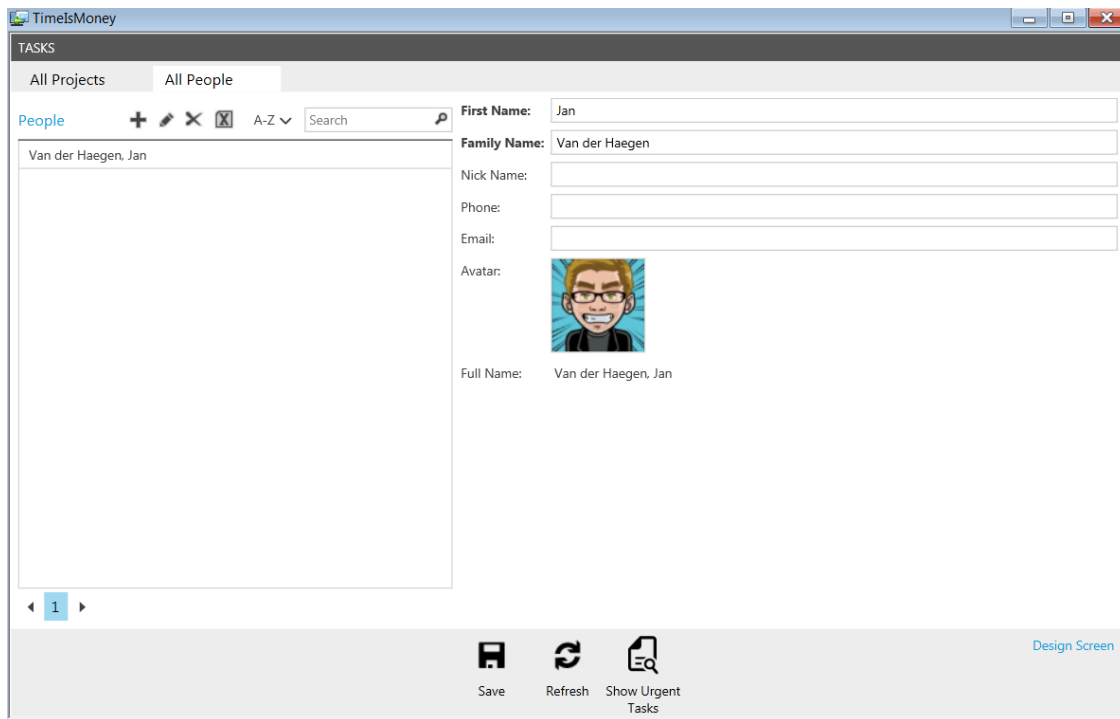


Figure 55: The Show Urgent Tasks command is now visible in the application

When the **Show Urgent Tasks** command is clicked, our new search screen called **My Urgent Tasks** opens.

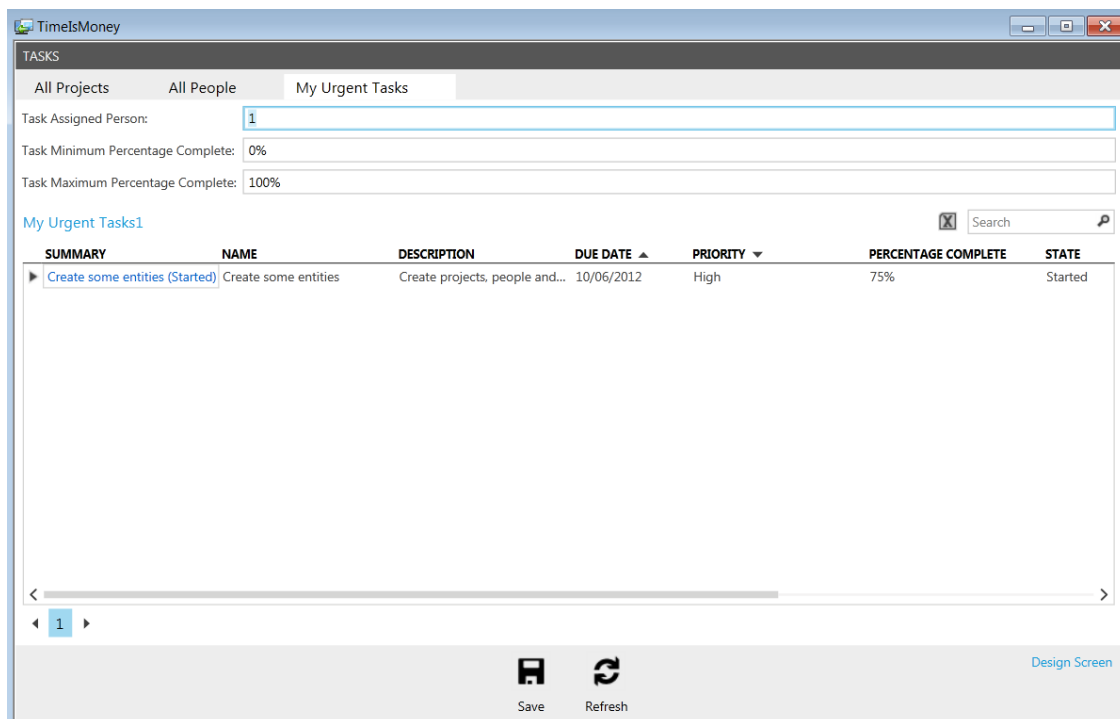


Figure 56: My Urgent Tasks screen

The run-time Screen Editor

Although our new screen is useful, you'll probably want to make some graphical modifications to it, like hide the **Task Assigned Person** label and text box. One of the most jaw-dropping features of Visual Studio LightSwitch is that you do not have to write custom XAML to make modifications to the UI if you do not want to or are not familiar with Silverlight XAML. Even better, you do not even have to stop running your application and recompile it if you want to change the layout or controls on your screens.

If you launch a LightSwitch application from the Visual Studio IDE (i.e. with the debugger attached), the application will have a command called **Design Screen**, which enables you to do these modifications and persist them back to your Visual Studio solution at run time! Quite honestly, it could be a personal lack of experience, but I do not know of any other technology that allows you to do this, and I'm loving it. Designing the graphical parts of your application while it's running without having to build to test each modification, and to be able to design with actual test data instead of "*Neque porro quisquam est qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit*"—This feature is an absolute time saver!

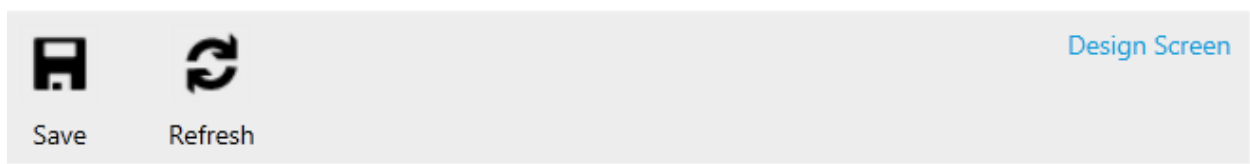


Figure 57: The Design Screen command

Click the **Design Screen** command in the application's command bar. This will open a run-time variant of the **Screen Editor**.

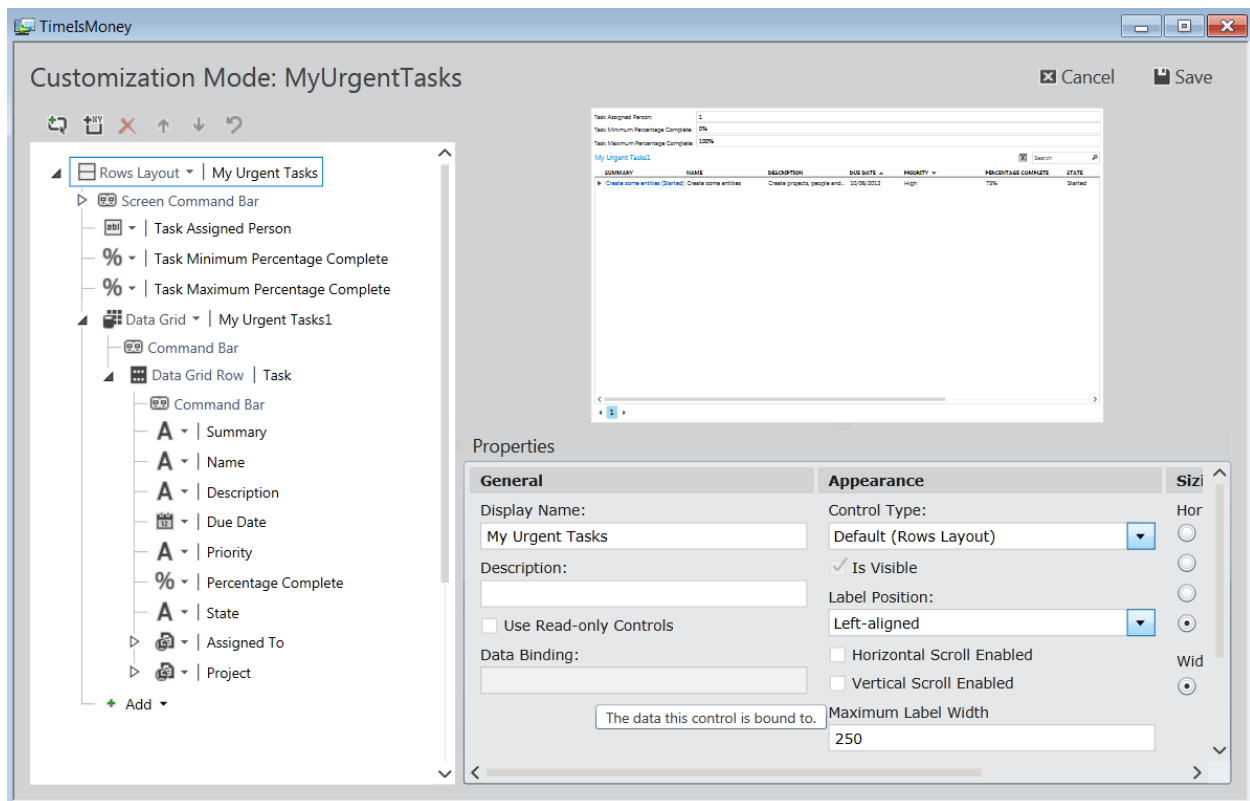


Figure 58: Using the Screen Designer at run time

Hiding the *TaskAssignedPerson* controls

In the **Visual Tree** editor on the left, select the **Task Assigned Person** element, and make sure the **Is Visible** check box is cleared.

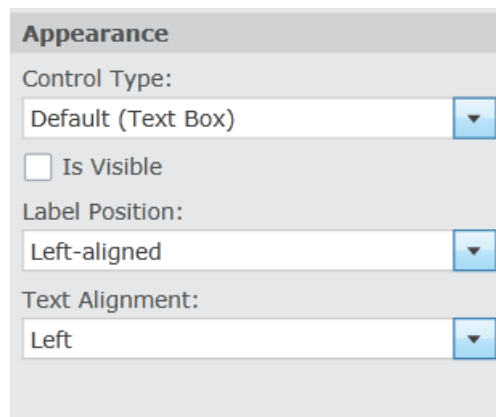


Figure 59: Designing the screen at run time

Once you click the **Save** button in the top right of your screen, the screen will be updated and your changes will be persisted back to the Visual Studio solution.

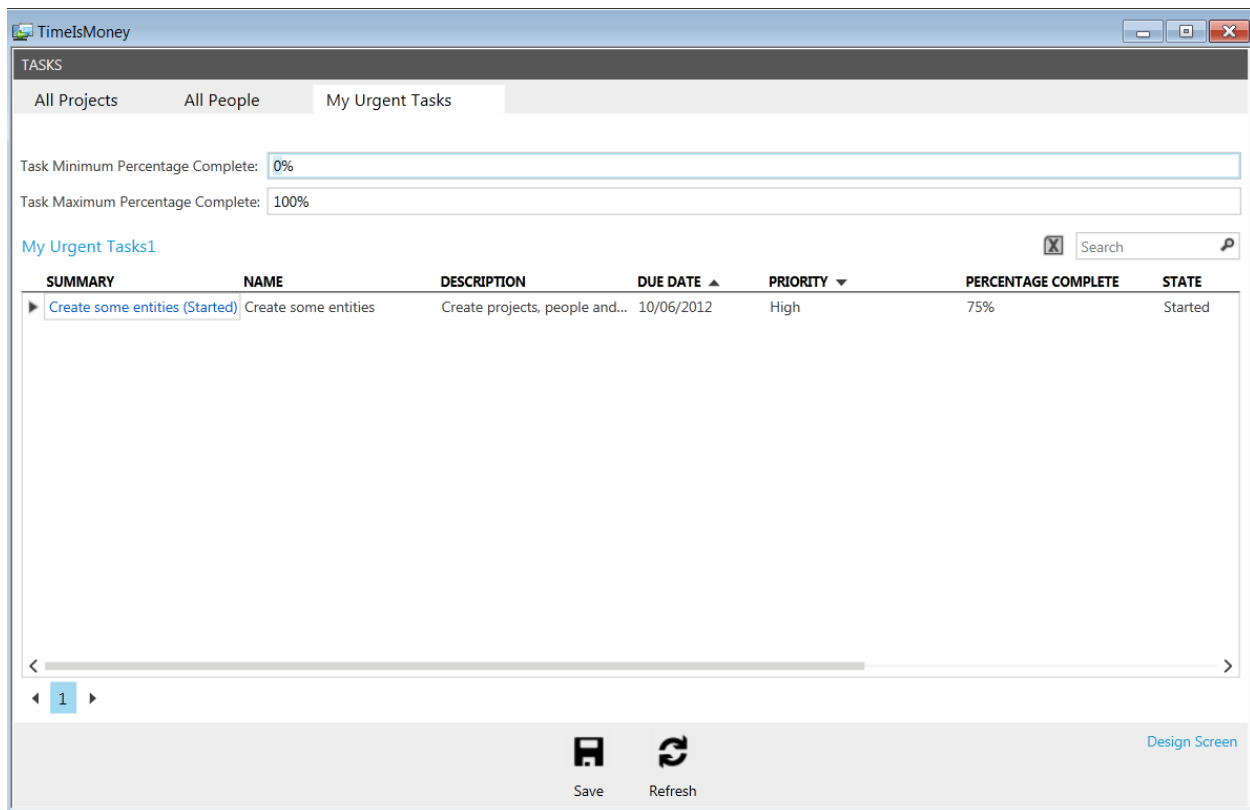


Figure 60: After editing the search data screen at run time

Creating new item templates

This run-time version of the **Screen Designer** can do more than set properties on existing visual elements. It can add new elements to your screen or replace existing elements with a completely different one. For example, in the run-time **Screen Designer** of the **All People** screen, select the **Person** visual element.

The control that is currently used is the **Summary Viewer**, which results in the following representation.

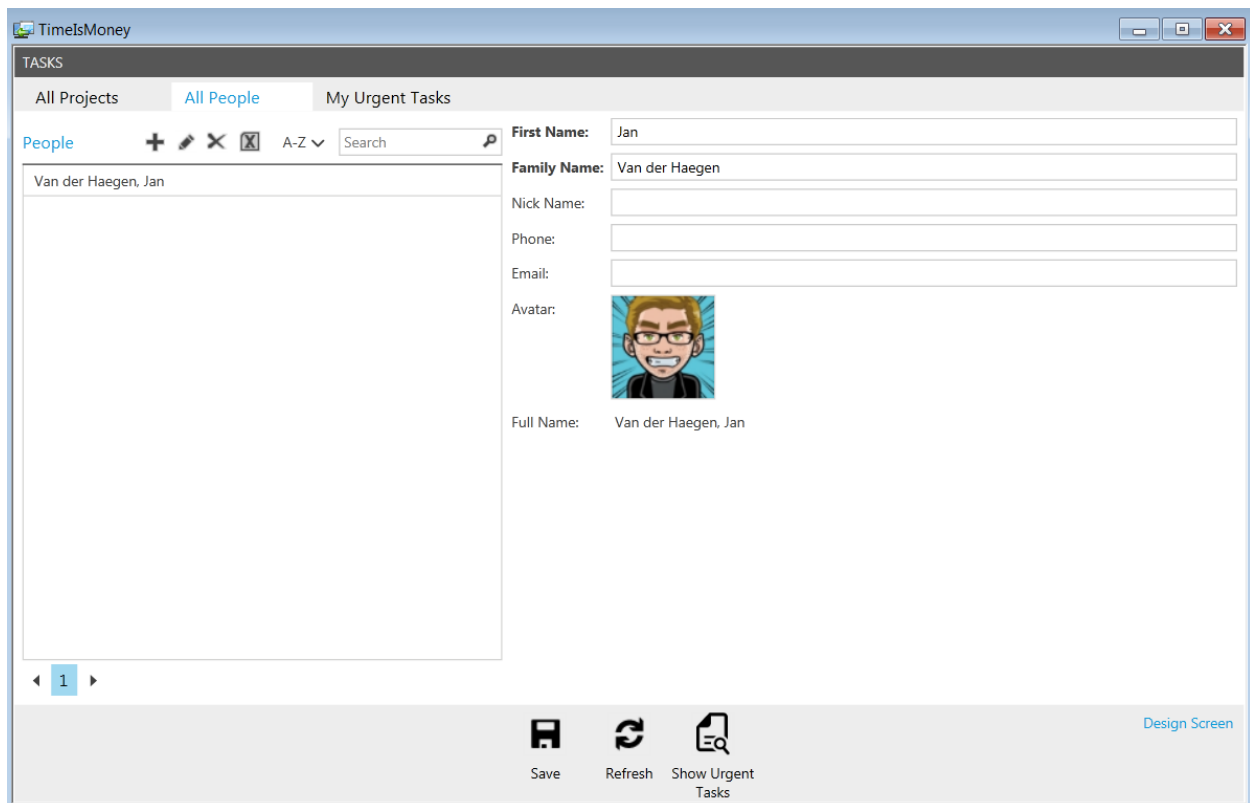


Figure 61: The current All People screen

By clicking on the down arrow, you can select an entirely different layout for this list's item data template.

Change the control used to represent a **Person** to **Picture and Text** by selecting it from the drop-down menu.

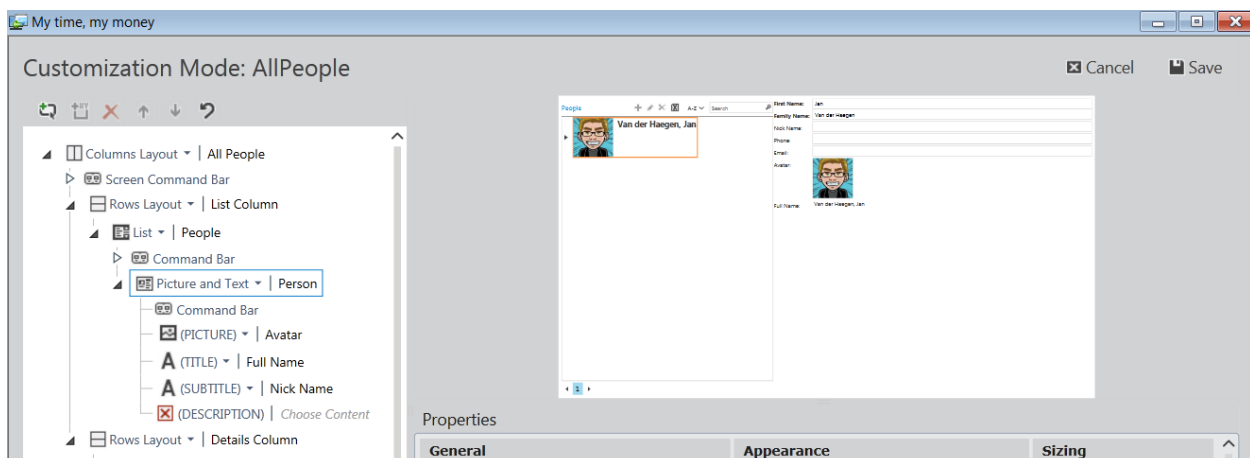


Figure 62: Changing the item template at run time

In this **Picture and Text** editor, use the **Avatar** property as the **Picture** and the **Full Name** computed property as the **Title**.

Modifying an entire group of controls at once

The run-time **Screen Editor** can also change properties for an entire group as well. Select the **Details Column**—it is the second **Rows Layout** visual element in the **Visual Tree**.

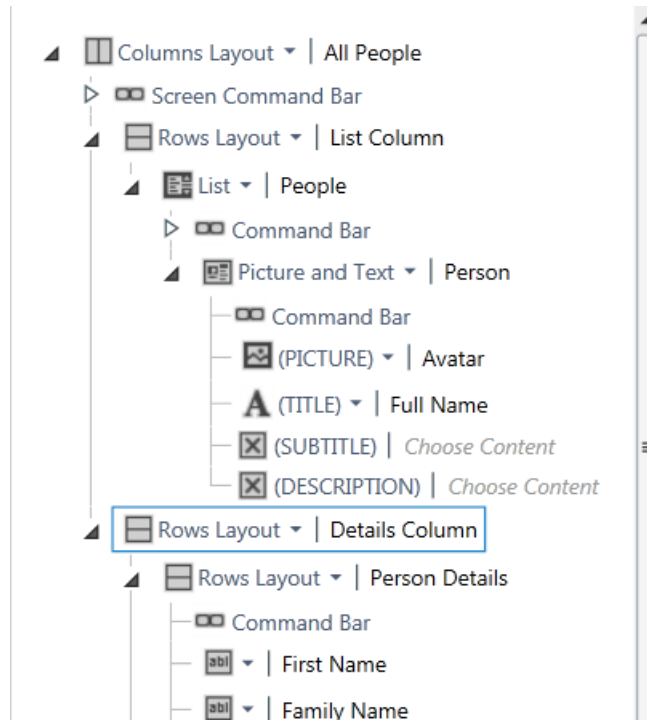


Figure 63: Designing an entire part of the screen at once

In the **Properties Window**, select the **Use Read-only Controls** check box as shown in the following figure, and then click **Save**.

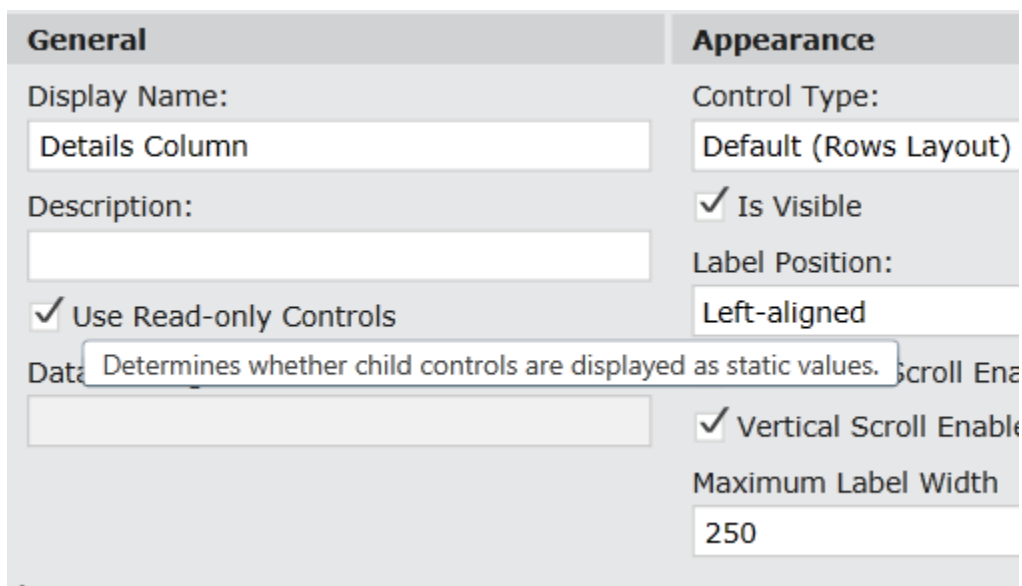


Figure 64: Setting the read-only status of an entire collection of controls at run time

The changes are persisted back to the Visual Studio solution, and the screen is refreshed and now looks somewhat more graphically appealing.

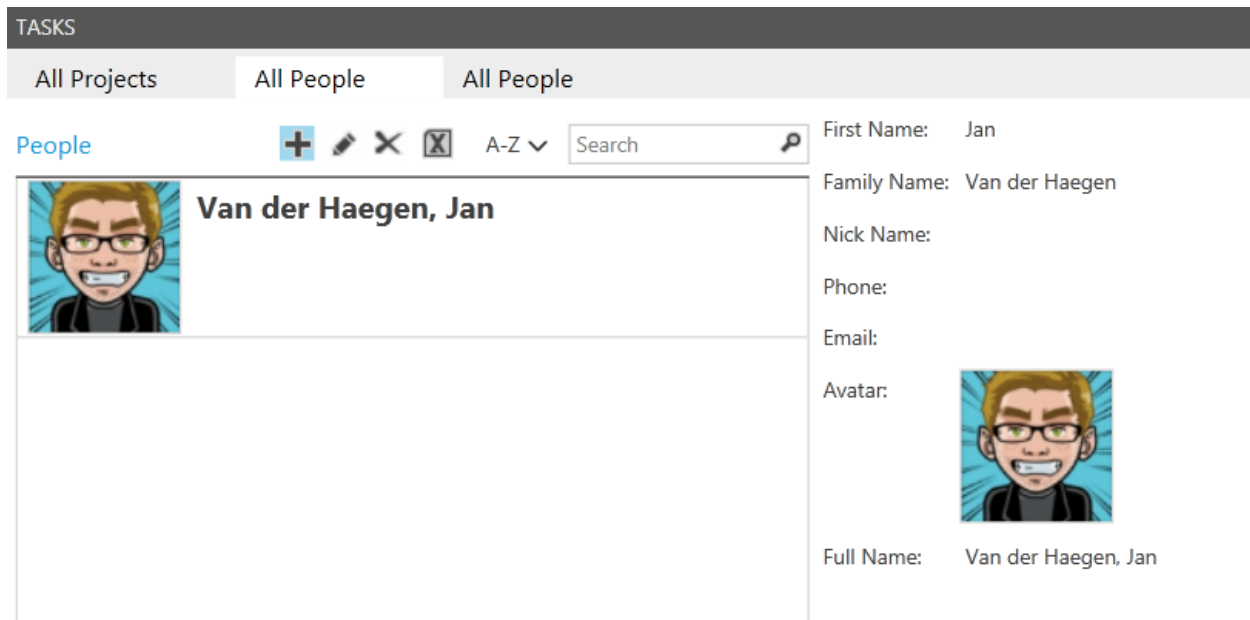


Figure 65: All People screen after designing it at run time

Just as you can write custom code to respond to a lot of predefined events in the application, you can also take full control of the UI and write custom Silverlight XAML for any element in the **Visual Tree Designer**.

However, because this requires rebuilding to compile the XAML, it cannot be done from the run-time version of the **Screen Designer**.

Chapter 6 Application Editor

Extending LightSwitch applications

Writing custom code or custom user controls is sometimes referred to as **level-one extensibility**. LightSwitch allows you to extend your application beyond what comes out of the box, and does not require you to have Visual Studio completely installed. Level-one extensibility is available in the stand-alone version of LightSwitch as well.

As this suggests, there are higher levels of extensibility as well. **Level-two extensibility** adds capabilities to a LightSwitch application that can be reused in different applications. They are distributed via Visual Studio Extensions (.vsix) and can be purchased directly from a vendor or downloaded from the Visual Studio Gallery

(<http://visualstudiogallery.msdn.microsoft.com/site/search?query=LightSwitch&f%5B0%5D.Value=LightSwitch&f%5B0%5D.Type=SearchText&ac=8>).

Traditionally, these extensions will add one or more of the following capabilities to your Visual Studio LightSwitch IDE or application:

- Reusable WCF services
- Reusable Business Types
- Reusable Controls
- Reusable Screen Templates
- Reusable Shells
- Reusable Themes

The final two, themes and shells, provide an easy way to change the entire look and feel (respectively) of your application with just a couple of clicks.

Installing an extension in Visual Studio

Close your LightSwitch application, and from the **Tools** menu in Visual Studio, select the **Extension Manager**.

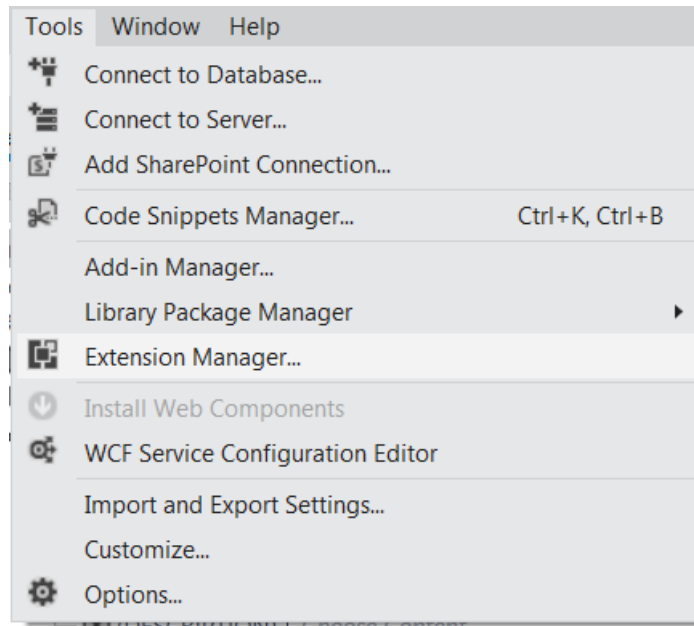


Figure 66: Opening the Extension Manager

In the **Extension Manager**, search the **Online Extensions** for a LightSwitch theme, shell, or both. Click the **Install** button, and then close the **Extension Manager**.

Activating an extension in your application

To activate your chosen shell or theme, select the LightSwitch project in the **Solution Explorer** and press Alt+Enter, or right-click on it and select **Properties** from the context menu.

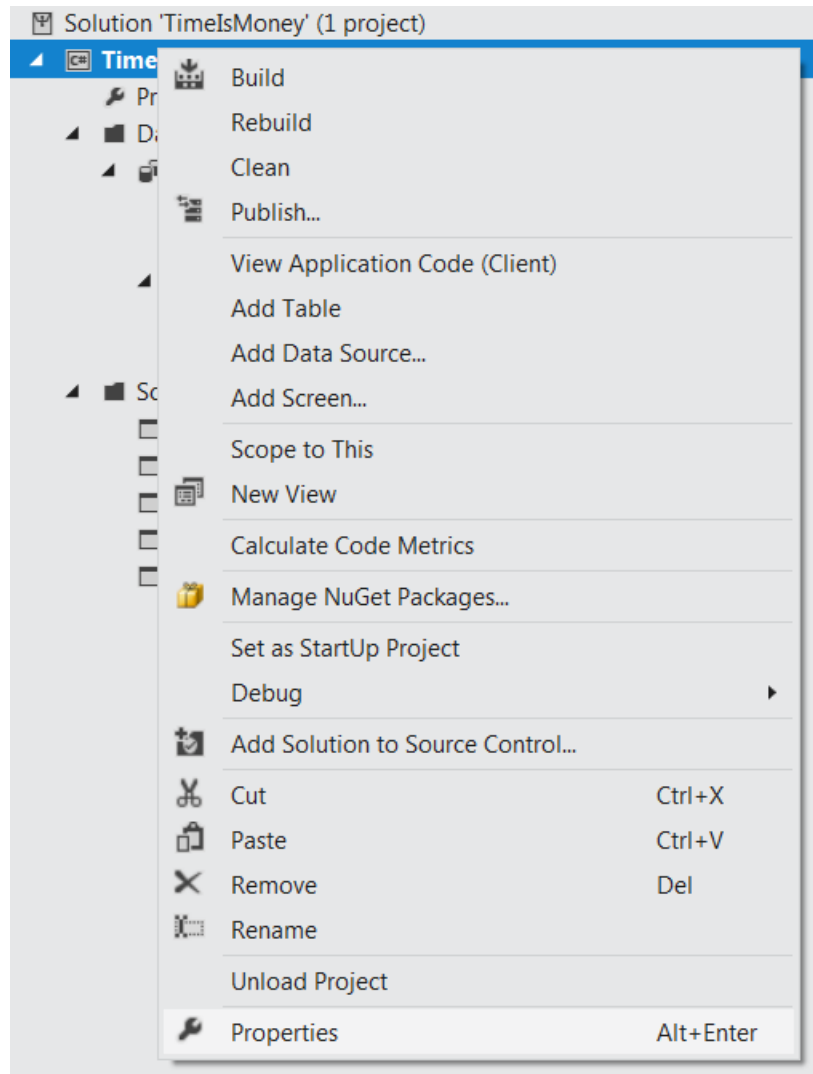


Figure 67: Opening the Application Designer

This opens up the fourth and final LightSwitch-specific designer, the **Application Designer**.

Navigate to the **Extensions** tab.

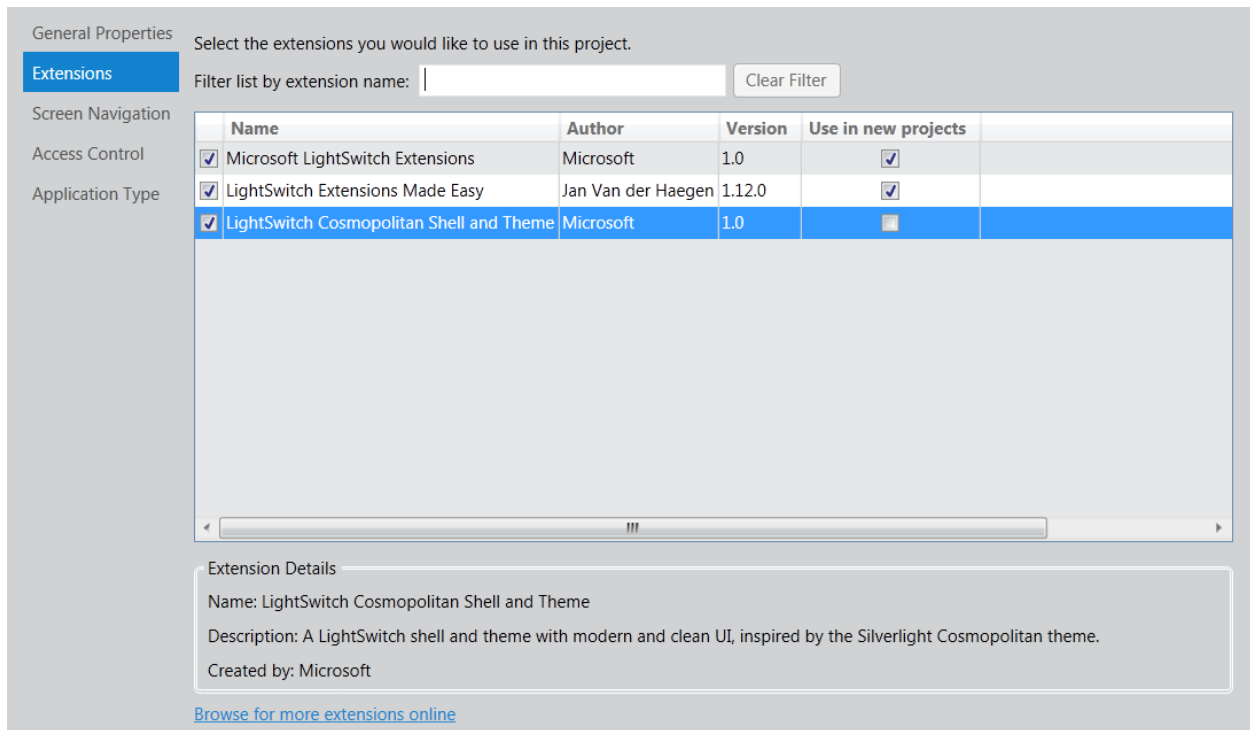


Figure 68: Activating an extension

In this tab in the **Application Designer**, you get an overview of all LightSwitch-specific extensions that you have available on your machine. Select the check box next to the extension that you have downloaded to activate it in this application.

In the previous figure, you can see I downloaded the **LightSwitch Cosmopolitan Shell and Theme**, which is now included in Visual Studio LightSwitch 2012 RC as the new default shell and theme.

The General Properties tab

Once activated, the level-two extensibility capabilities—the shell and the theme—can be used in this LightSwitch application. Selecting which shell or theme is used is done from the **General Properties** tab.

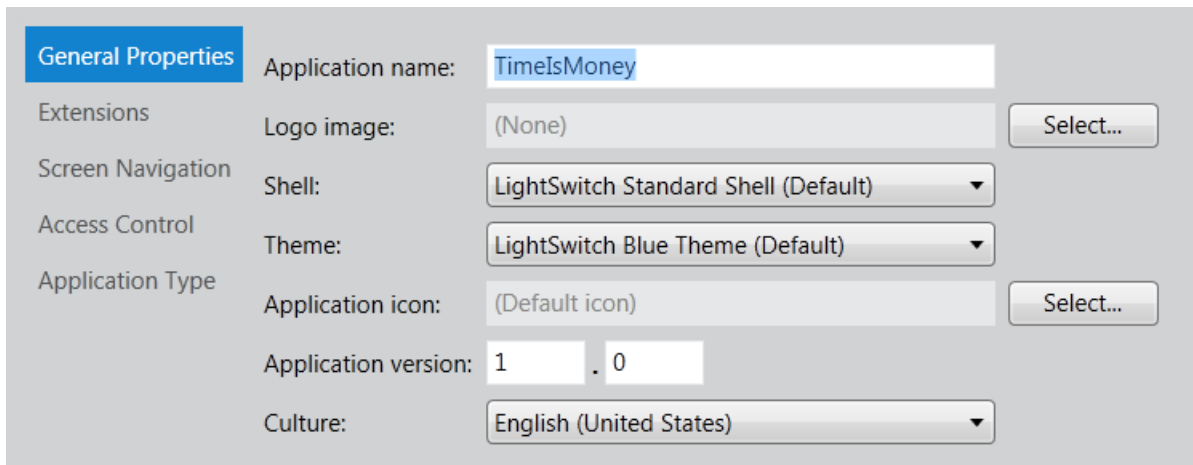


Figure 69: The Application Designer's General Properties tab

Change the selected **Shell** to the now available shell, and/or the selected **Theme** to the theme that you downloaded.

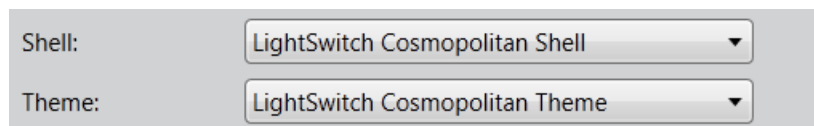


Figure 70: Selecting a shell or a theme to use in the LightSwitch client application

The other properties on the **General Properties** tab are worth a quick review as well:

- **Application name:** The name of the application, I changed it to *My time, my money*
- **Logo:** The logo to use if the shell supports it (which the **LightSwitch Cosmopolitan Shell** does).
- **Icon:** The icon of the application.
- **Version:** The application's major and minor release version. A third number, the revision number, is appended each time you publish the application with the same major and minor release version.
- **Culture:** The culture of the application. This is used for globalization and translations throughout the application. At the moment, LightSwitch only supports one culture at a time.

Press **F5** to build and debug the application, and see how the entire look and feel of your application has changed.

Screen Navigation

The **Screen Navigation** tab allows you to take control of the navigation menu of your application. Screens can be grouped in different groups, and the order and labels can be changed with this editor.

Navigation groups allow you to define the menus for your application. You can create additional navigation groups and assign screens to groups.

You can hide or show menus for users in different roles by editing the <Screen>_CanRun method in the application code:

[Click here to view application code](#)

Navigation menu structure:

The screenshot shows a tree view of the navigation menu structure. The 'Tasks' group is expanded, showing sub-items: 'All People', 'All Projects', 'All Tasks', 'All Urgent Tasks', and 'Include Screen'. The 'Administration' group is also expanded, showing sub-items: 'Roles', 'Users', 'Include Screen', and 'Add Group'. The 'Current startup screen' is set to 'Tasks / All People'. There are 'Set' and 'Clear' buttons at the bottom.

Figure 71: Exploring the Screen Navigation tab

You might have noticed that there's an **Administration** group, which wasn't visible at run time.

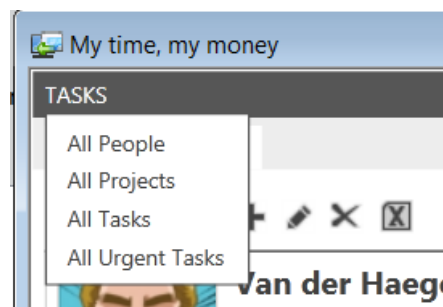


Figure 72: The LightSwitch application does not have an Administration group like the Navigation Tab suggested

The reason is twofold: We haven't activated a **User Authentication Mode** yet, and we do not have the correct permission to access these screens yet. Both can be changed from the **Access Control** tab.

The Access Control tab

The fourth **Application Editor** tab provides control over the authentication mode, allows you to define permissions, and allows you to force a secure connection for your OData endpoints (see [Chapter 3](#)).

The three supported authentication modes are:

- **Windows authentication:** Automatically log in using your Windows credentials. Advised for applications that run within a company.
- **Forms authentication:** Start the application with a login screen where users have to supply their username and password. Advised for applications that have users from different companies.
- **None:** Start the application and require no authentication. This authentication mode is never advised unless you're the only user and are running the application on your local machine.

General 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.

Extensions

Screen Navigation

Access Control

Application Type

Select the type of authentication to use:

☐ 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

☐ Do not enable authentication

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.	<input checked="" type="checkbox"/>
<Add New Permission>			<input type="checkbox"/>

Require Secure Connection (HTTPS)

This setting protects the end-users' credentials and application data by requiring a secure, encrypted connection when accessing the server.

☒ Off. HTTPS connection is not required.

☐ On. Users must connect using HTTPS (must be configured on the server).

Note that when this setting is on, the website must be properly configured to use HTTPS.

[Click here to find out more.](#)

Figure 73: The Application Designer's Access Control tab

For this application, change the **Authentication mode** to **Use Forms authentication**. When the application is run from Visual Studio, i.e. with the debugger attached, LightSwitch will automatically skip the login screen and log in with a user called **Test User**.

When it comes to authorization, LightSwitch implements a permission-based system. These permissions are defined in this tab of the **Application Designer**, and later checked anywhere

you need to via custom code. This can be done on a screen level, entity level, or even operation level.

By checking the **Granted for debug** check box (see Figure 73) next to the only predefined permission, called **SecurityAdministration**, you are adding this permission to the **Test User** that will be used when debugging. Press F5 to build and start debugging, and notice the difference.

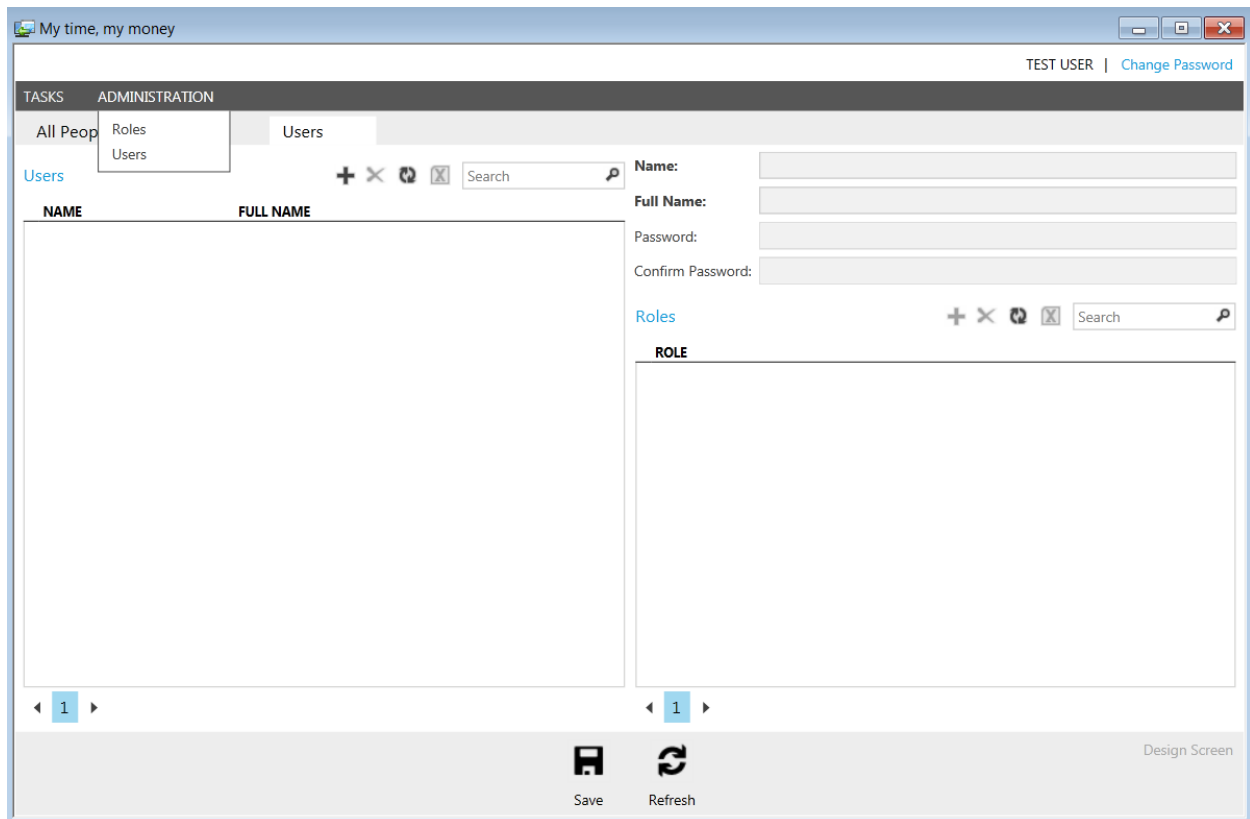


Figure 74: After setting the access control properties, the Administration menu is visible

For those of you who do not like “find the 7 differences” games, notice the **CurrentUserControl** is now visible at the top right to show that LightSwitch has indeed logged in a **Test User** automatically—another great time saver.

This **Test User** also has the **SecurityAdministration** permission, which is required to see the **Administration** group in the navigation menu. This group contains two screens where a security administrator can manage the users and their permissions. Note that your test data is not deployed when you deploy your application, including these users and their permissions, so changes made via these screens during development will not be reflected in your deployment. These screens will really only become useful after deploying your application.

Speaking of which...

The Application Type tab

Time for the final jaw-dropper, the pièce de résistance, the climax of our symphony. We started this e-book by referring to Visual Studio LightSwitch as “the easiest way to build data-centric applications for the desktop or the cloud,” however throughout the tutorial, with the exception of mentioning the public OData endpoints of the server in [Chapter 3](#), we haven’t really stated anything about the architecture or target platform of the application. So what did we build? A single-tier desktop application, a multi-tier one, or a web application that can be hosted in the cloud?

Because of the metadata-driven implementation of LightSwitch, the final tab of the **Application Designer**, called the **Application Type** tab, allows you to make these decisions at the latest possible stage of development. This offers more application flexibility than the traditional development cycle in which you have to decide your application deployment environment at the very start. LightSwitch’s flexibility also allows you to change the application type without having to rework entire parts of your application.

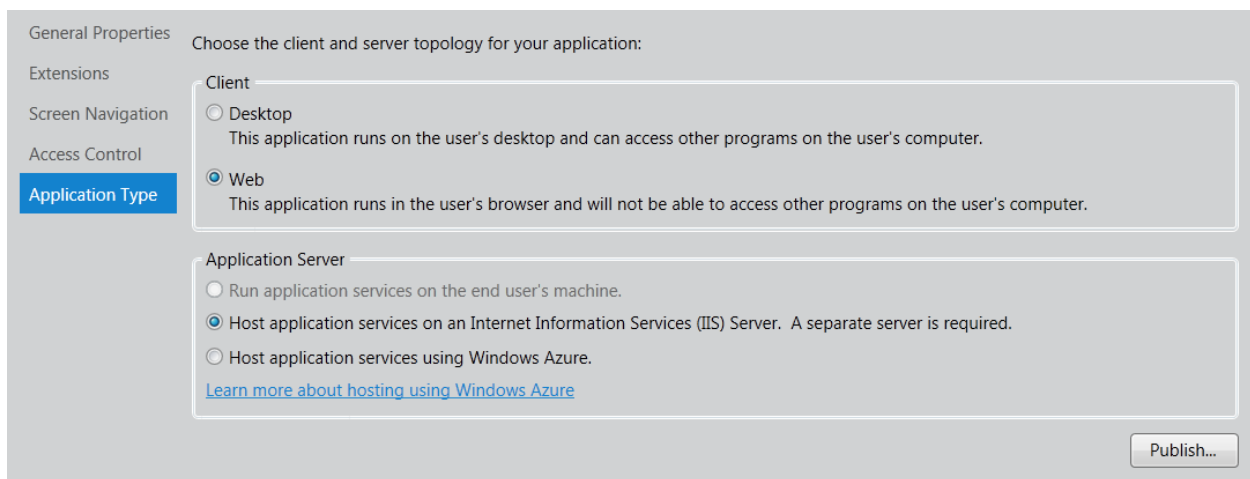


Figure 75: Application Designer's Application Type tab

Think about the latest large data-centric application you worked on. Consider that the application was originally written as a desktop application that directly accessed the database (single tier), and after some time a decision was made to refactor it to a web application that called web services (multi-tier). How long would it have taken to refactor the application?

Here comes the LightSwitch version of that story.

With one click, change the **Client** type to **Web**. This automatically disables the choice to run the **Application Server** on the same tier as the client. Press F5 to build and start debugging.

During the build process, LightSwitch will take the metadata that you have designed in the **Entity Designer**, along with any that you have designed in the **Query Designer**, the **Screen Designer**, and all the choices you have made in the **Application Designer**, to generate a multi-tier web application.

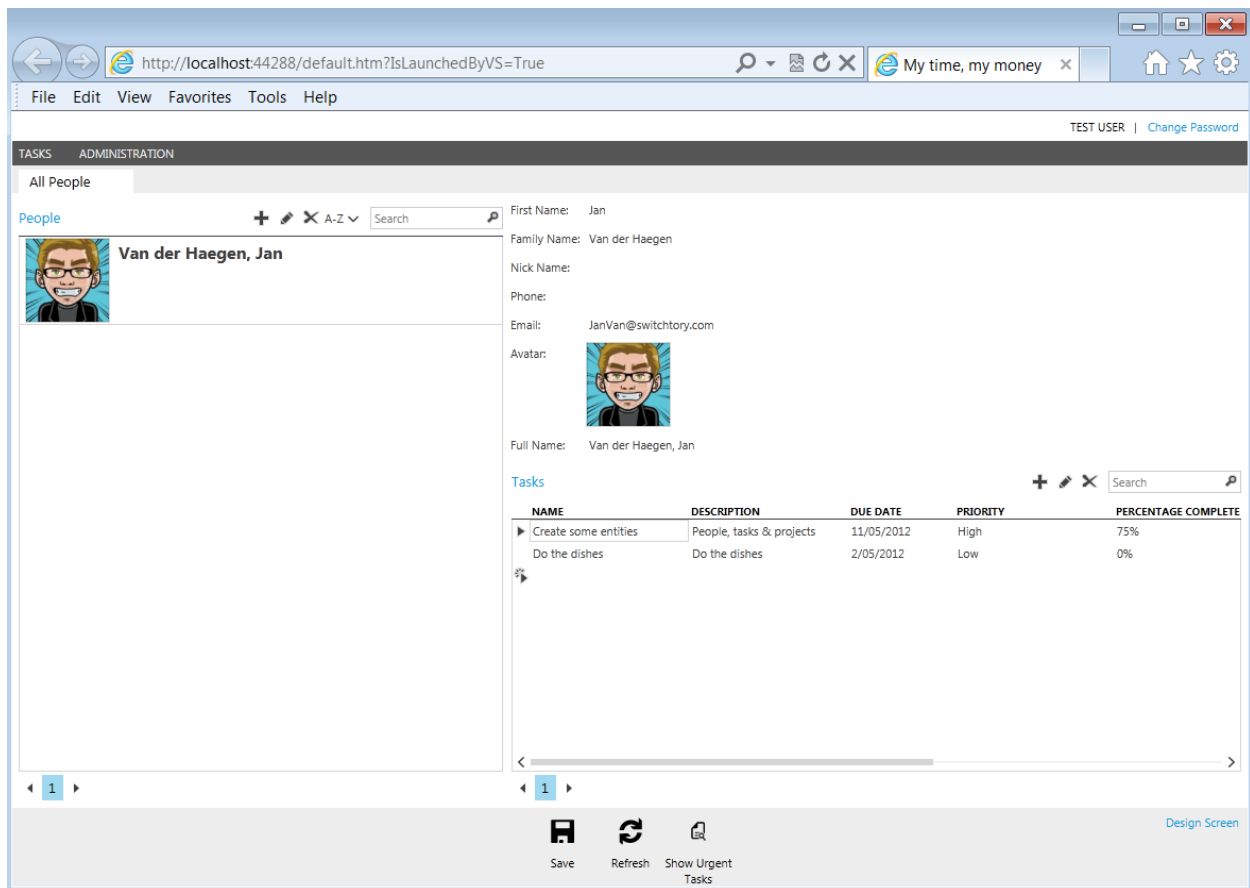


Figure 76: Changing a single-tier desktop application to a multi-tier web application with one simple click

Publishing time...

If you are satisfied with the result, click the **Publish...** button in the bottom right of the **Application Tab** in the **Application Designer**, and start dreaming about a lucrative subscription model for your freshly created application.

Chapter 7 Moving On

Is Visual Studio LightSwitch the right tool?

During the past year, I spent quite some time doing research on the LightSwitch product and my journey in its metadata-driven implementation has been a life-altering experience. LightSwitch 2012 RC uses a newer version of Silverlight on the client tier and an entirely different technology on the server tier (which also opens up the application to create clients in any other technology) than its predecessor, and the fact that my LightSwitch 1.0 applications can be converted without any modifications, a refactoring battle that would normally take years to complete, proves to me that LightSwitch does something in application development more “right” than I could ever dream of.

In one IDE, citizen developers, professional developers, and expert developers come together to create the best data-centric applications in the shortest amount of time.

Citizen developers, traditionally people with limited technical knowledge or coding skills but who know and understand business problems like no others, can use simple yet powerful editors to design an application without ever suffering from hitting a brick wall, something most rapid application development IDEs suffer from, thanks to the numerous level-one extensibility points.

Professional developers can stray away from the tasks they find boring because they feel it doesn’t challenge them, and focus on what they do best: delivering reusable capabilities in the form of level-two extensions: reusable control suites, business types, shells, themes, etc.

The Microsoft LightSwitch teams, as expert developers, make sure the characteristics of citizen and professional developers blend together in powerful line-of-business applications that follow the best and industry-accepted technologies and architectural trends.

If the software challenge at hand has anything to do with data, and that’s the biggest “if” when considering whether LightSwitch is a suitable tool, I’ll always be eager to use LightSwitch to solve it. And not just because of the blazingly fast results you can get with the product, but for the enjoyable development experience as well.

Learning more about Visual Studio LightSwitch

If this e-book has even slightly convinced you to learn more about Visual Studio LightSwitch, there are a couple of great places to get started.

- Microsoft’s official LightSwitch Developer Center (<http://msdn.microsoft.com/en-us/lightswitch/default>) is a great place to get started. Beth Massi (www.BethMassi.com), nicknamed “the LightSwitch goddess” by the community, is the most publicly visible senior LightSwitch program manager and has created numerous “getting started” videos and tutorials. New content of both introductory-level and expert-level topics is added at least on a weekly basis. A great monthly community content “rollup” is posted that has links to MSDN forums where LightSwitch team members will help you with any question or problem you might have if other community members haven’t already beat them to it.

- Michael Washington, one of the LightSwitch community's greatest rock stars, hosts the LightSwitch help website (<http://www.LightSwitchHelpWebsite.com>). This unofficial community portal offers a lot of great links, a forum where the community members help each other out, and a marketplace where anyone can sell his or her services and extensions. Michael frequently treats visitors to new innovative articles intended more for experts than novices.
- You'll find many interesting, privately held blogs by other rock stars in the LightSwitch community scattered throughout the Internet. Representing the different backgrounds of the authors, these blogs often discuss very different aspects of LightSwitch, or approach it from a very different angle. My personal blog, for example (<http://janvanderhaegen.wordpress.com>), focuses solely on the internals of the framework, whereas Paul Patterson's blog (<http://www.paulspatterson.com/>) shows some amazing craftsmanship that even citizen developers could pull off. The [image](#) in the introduction of this book shows a LightSwitch application called A Little Productivity which he created and deployed to Azure in less than one week—an experience that he has blogged about in depth.
- I also created a bot that monitors different official and unofficial sources, including the blogs of some of the most experienced community members like Yann Duran, Jan D'hondt, The SD Times, Dan Beall, Paul Van Bladel, Keith Craigo, Kostas Christodoulou, Stu (stuxstu), Regan Ashworth, Allesandro del Sole, Michael Washington, Bala (Tek Freak), Tim Leung, Paul Patterson, Jewel Lambert, and Rashmi Ranjan Panigrahi. Whenever any interesting LightSwitch material pops up on the web, the bot tweets under the [@LightSwitchNews](#) account. If you feel anyone is missing from this list, I must apologize as it is a sign of ignorance on my account. Feel free to let me know and I'll gladly make the bot up to date.