THE EXPERT'S VOICE® IN .NET

Visual Studio LightSwitch 2012

CREATE BUSINESS APPLICATIONS EASILY, QUICKLY, AND EFFECTIVELY

Tim Leung

Apress®

For your convenience Apress has placed some of the front matter material after the index. Please use the Bookmarks and Contents at a Glance links to access them.

Apress®

Contents at a Glance

Introduction

It's possible to see many things in life as a journey, and writing a business application is no exception. On this particular journey, your goal is to build an application that fully works and meets the expectations of your users. Let's imagine, for a moment, that the tool that builds your application represents your journey's mode of transport. Using this analogy, I like to think of LightSwitch as a high-speed train because it allows you to reach your destination a lot faster than usual.

Speaking of trains, journeys are much more comfortable today than they were in the past. Modern carriages are Wi-Fi enabled and include tables and electrical sockets that you can plug your laptop into. There are even touch-screen displays that you can use to watch television or to view a map that shows you exactly where you are, and how much further there is to go. The ride itself is smooth enough for you to leave your hot coffee on the table without fear of it spilling. Everything is much cleaner than in the age of steam; and you never risk getting a hot cinder in your eye if you open a window!

The fascinating thing is that your railway journey might follow exactly the same route as it would have 150 years ago. However, the experience itself is quicker, cleaner, and more convenient. Just like the railways, LightSwitch has evolved during its short lifespan. The improvements in the latest version help to keep it fresh, relevant, and purposeful.

When LightSwitch first came out, it created applications based on Microsoft Silverlight. Today, you can support mobile and touch-screen tablet devices by extending your application to include an HTML front end. A big benefit of this approach is that you can reuse all of your existing data and business logic. This technical change is the railway equivalent of swapping all the carriages on a train for modern replacements.

Likewise, the first version of LightSwitch employed RIA services as the communication channel between the client and server. Today's LightSwitch uses the OData protocol instead. Applying the railway analogy once more, this is like replacing the railway tracks with smoother rails. If LightSwitch doesn't totally meet your requirements, you can easily integrate your application with other systems and platforms—like completing the final leg of the journey by car, bus, or foot.

The point of this analogy is to highlight one of the beauties of LightSwitch. It's possible for LightSwitch to evolve and improve because it's built in a modular way that allows individual pieces to be replaced. This modular architecture ensures that LightSwitch will remain relevant for a long time to come.

You can also consider this book as a journey—a tour that shows you how to build a LightSwitch application from start to finish. As with most journeys, there'll be highlights and features along the way. I'm really pleased that you've chosen to take the LightSwitch journey. It's sure to save you time and make your life easier; so I'm certain it'll be a journey that you won't regret taking!

Who This Book Is For

This book is designed for readers who want to build data-driven business applications quickly. You don't need to have any prior knowledge of LightSwitch, but it helps if you know a little about .NET and database development in general.

How This Book Is Organized

If learning LightSwitch seems like an enormous task, don't worry! You might have heard of the Pareto Principle, which also goes by the name of the 80/20 rule. This rule states that in most situations, 20% of something causes 80% of the results.

If you apply the Pareto Principle to this book, it suggests that you can accomplish 80% of what you want to accomplish by reading less than four chapters! And, in fact, the first three chapters highlight the key topics that are all you need to build an application that's almost 80% complete.

By the end of Chapter 3, you'll understand how to create data tables and attach to external data. You'll know how to build a Silverlight web or desktop application that includes screens and navigation controls. You'll see how to open child records from a parent record and how to assign parent records to child records by using controls like the autocomplete box (a picker control that's similar to a drop-down box). In short, you'll know how to build a working business application with enough functionality to perform most basic tasks.

The next part of the book shows you how to write code and queries. After that, you'll learn how to create screens that contain advanced features and support mobile devices by adding an HTML client.

Unless you want to create an application that's completely isolated from other systems, it's useful to know how you can attach to unusual data sources and how to share your LightSwitch data with other applications. You'll discover how to do this through RIA services and OData.

You can extend your Silverlight applications through custom controls and extensions, and you'll find an entire section of the book devoted to this topic.

The remaining chapters of the book show you how to build reports, integrate with e-mail systems, and deploy your application.

The Sample Application

Most of the chapters include code samples that refer to a HelpDesk application. The purpose of this application is to relate the technical content in the book to a real-life scenario that you can understand.

The HelpDesk application is typical of the applications that you can build with LightSwitch. Its features are simple enough to understand, yet complex enough to show off the more advanced features of LightSwitch. It is designed to help companies manage problems, and it's especially suitable for departments that deal with technical support issues. It allows users to record the actions they carry out to resolve a problem, while giving managers an overview of all current issues.

Figure 1 shows a screen from this application and illustrates some of the features that you'll learn about in this book.

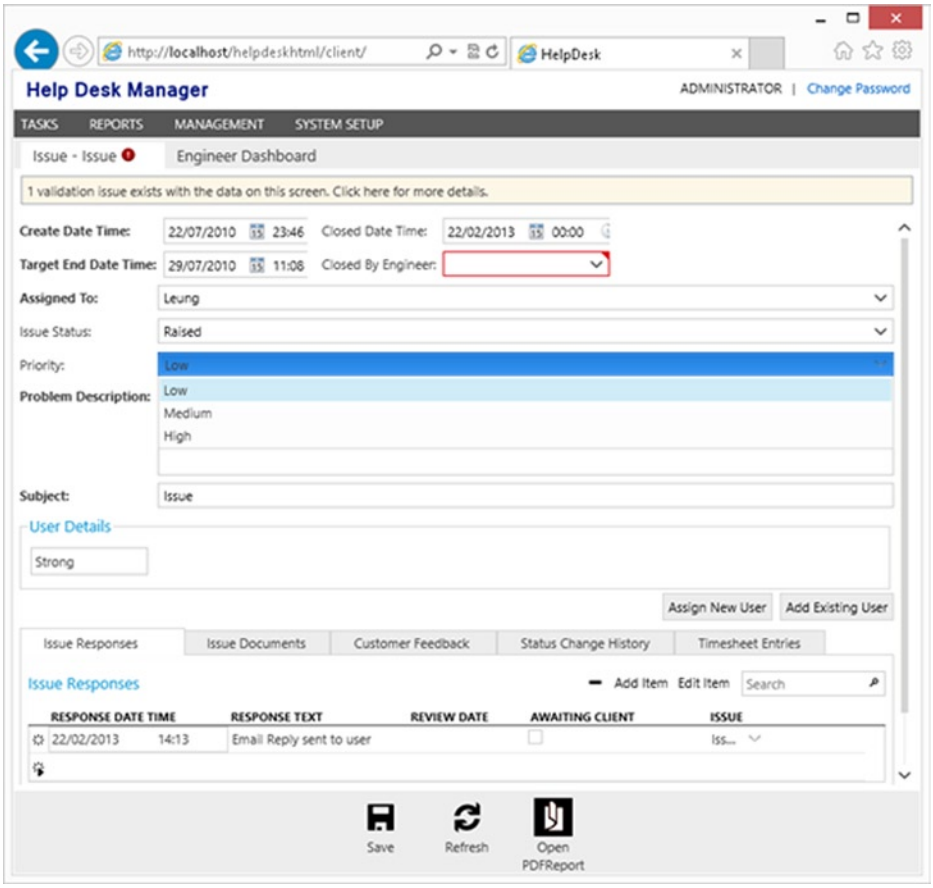

Figure 1. *A typical screen from the HelpDesk manager*

Figure 1 shows the data-entry screen through which users can enter help desk issues. They can set the user that the issue relates to and allocate an engineer to the record. The data-entry screen allows users to respond to the issue, attach documents, register feedback, and record the time that engineers spend on the task. To help you re-create this application, you can find a summary of the application's tables in Appendix E.

Code Samples

LightSwitch supports C#, VB.NET, and JavaScript. This book includes code samples in all three languages. To make life simple, LightSwitch hides much of the complexity that's associated with application design. When you want to write some code, you do this by clicking on a *write code* button that LightSwitch shows in its graphical designers. The write code button shows a list of events that you can handle. When you select one of the options in the list, LightSwitch opens a code editor window that you can use to author your code.

When you're starting out with LightSwitch, it isn't always obvious where your code is stored. To add some clarity, each code listing includes a file location, as shown in Listing 1.

Listing 1. Hello World Example

VB:
File: HelpDeskVB\Server\UserCode\ApplicationData.vb

```
Imports System.Text.RegularExpressions

'REM VB Code appears here                                    ❶
Dim message = "Hello World"                                  ❷
```

C#:
File: HelpDeskCS\Server\UserCode\ApplicationData.cs

```
using System.Text.RegularExpressions;

//REM C# Code appears here                                   ❶
var message = "Hello World";                                 ❷
```

For both the VB and C# examples in Listing 1, the `File` heading specifies the file name and path. In this example, `HelpDeskVB` and `HelpDeskCS` refer to the name of your LightSwitch application. The name Server refers to the Server project. This piece of information is useful because it informs you that the code runs on the server rather than the client. (You'll learn all about this in Chapter 1.) The next part of the file heading, `UserCode`, indicates a subfolder in the `Server` project. Finally, `ApplicationData.vb` and `ApplicationData.cs` refer to the name of the VB or C# file that the listing shows.

If a piece of code requires you to reference a .NET namespace, the code listing will show the necessary `Imports` (VB.NET) or using (C#) statement. Be sure to add these statements to the top of your code file.

Many of the code samples include numeric symbols to help you locate specific lines in a code listing. For example, line ❶ defines a comment, whereas line ❷ declares a variable. There's obviously no need to enter these numeric symbols when you re-create this code for real!

As in all books, the length of a code line can exceed the width of this book. In most cases, I've put line breaks in places that still allow the code to compile. But in some circumstances, this isn't always possible. Notable examples include namespace declarations in XAML files and VB.NET keywords like `inherits` and `implements`. If your application doesn't compile, it's worth checking your code for extraneous line breaks.

Tips/Notes/Cautions

This book includes callouts with tips, notes, and cautions. Tips are helpful hints or pieces of information that may be particularly interesting. Notes provide you with nonessential, yet interesting additional information about a particular topic. Cautions alert you to anything that might be detrimental to your work or could cause damage.

Exercises

Some readers prefer to learn kinesthetically (that is, in a touchy-feely way). If you fall into this category, you can try the exercises that you'll find throughout this book. These exercises provide you with ideas to try out in LightSwitch to help you further understand the content that's described in the book.

Comments and Errata

Although my editors and I have tried to be as accurate as possible, mistakes do sometimes happen (particularly with a book of this length). If you have any feedback or want to report any bugs, please visit the official page for this book at the Apress website:

http://www.apress.com/9781430250715

This page also shows any mistakes that we've found following the publication of this book.

CHAPTER 1

Introducing LightSwitch

Do you ever feel that software development is too complicated? If you're building business applications already, do you feel frustrated by the amount of time that it takes to build an application? Or if you've never built a modern-day web application, does the thought of doing so frighten you? If you're starting from scratch, where exactly do you begin? And are you really confident that you can build a robust application that meets the expectations of your users and isn't susceptible to "falling over" or getting hacked?

If any of this sounds familiar, LightSwitch will come as a breath of fresh air. It's a rapid software development tool that helps you to work faster, better, and smarter. It allows you to build data-centric business applications that can run on the desktop or on the web.

But despite LightSwitch's simplicity, there's no need to compromise on functionality. You can build applications with all the features that you'd expect to find in a typical business application, and this book will show you exactly how.

The first chapter teaches you the core LightSwitch principles that will help you throughout the rest of this book. The key topics that you'll cover in this chapter include

- The three-tier architecture that LightSwitch uses
- How LightSwitch uses a model-centric architecture, and applies the MVVM pattern
- What exactly happens when you build and compile a LightSwitch application

Let's begin by taking a look at LightSwitch's architecture.

Understanding LightSwitch's Architecture

In keeping with best practices, LightSwitch applications consist of three layers. This architecture enforces the logical separation of your application and provides better maintainability and scalability. The biggest advantage is that it allows Microsoft to rewrite or replace entire layers in response to changes in requirements or technology. Every time you create a new application, LightSwitch creates multiple projects in your LightSwitch solution. Each project is designed to run on a specific layer in this architecture.

This automation of the project layout itself is a major timesaver. If you built an application without LightSwitch, you could spend days on getting the architecture correct.

Figure 1-1 illustrates the parts that make up a LightSwitch application. It highlights the way that applications are separated into data, logic, and presentation layers. It also illustrates how the layers communicate with each other.

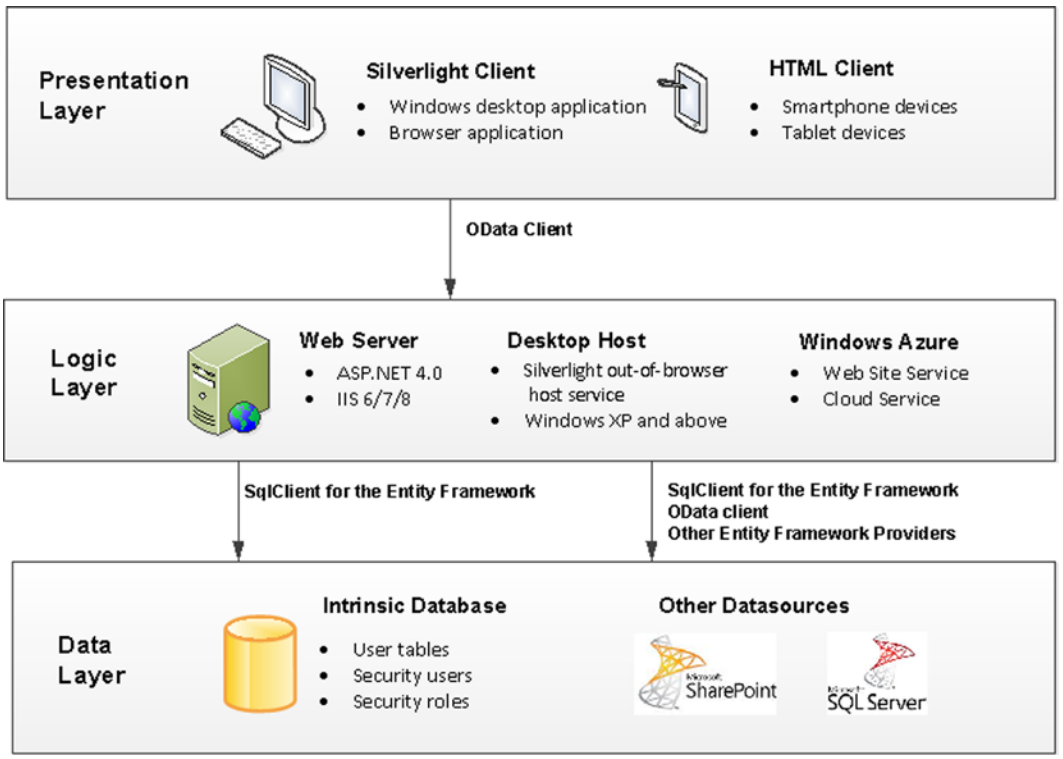

Figure 1-1. *LightSwitch's architecture*

The Data Layer

LightSwitch's architecture defines a separate logical data layer. This means that LightSwitch stores your data separately from the rest of your application, and allows you to run the data-storage processes on a platform that's separate from the rest of your application.

The advantage of this *data independence* is that it doesn't tie you to a specific database or datasource. If you're not happy with using Microsoft SQL Server, for example, you can more easily configure your application to store its data in Oracle instead.

Delegating the responsibility of data storage to a separate layer also improves scalability and performance. It allows you to take advantage of database-engine features such as mirroring, clustering, and indexes, and you can also upgrade your database engine without affecting the rest of your application.

In practical terms, LightSwitch allows you to read and write data from SQL Server, SQL Azure, SharePoint, and many other data sources. In fact, your data-storage options are almost limitless. This is because you can connect to almost any data source by creating your own custom RIA (Rich Internet Applications) service.

A special database that LightSwitch uses is the Intrinsic (or Application Data) database. LightSwitch uses this database to store the tables that you create through the built-in table designer.

LightSwitch allows you to define users and to apply security access control. To manage your users and roles, it uses the ASP.NET SQL membership provider. This membership provider relies on database tables and, by default, LightSwitch adds these tables to your Intrinsic database.

During design time, LightSwitch hosts your Intrinsic database using LocalDB (the successor to Microsoft SQL Server Express). But when you deploy your application, you can host your Intrinsic database on any version of SQL Server 2005 or above.

The Logic Layer

LightSwitch's architecture defines a separate logic layer that contains application services. A big advantage of this separation is that it allows Microsoft to rewrite parts of LightSwitch without impacting either the client or data tiers.

The application services are responsible for carrying out business rules, validation, and data access. LightSwitch creates a *Data Service* for each data source in your application and exposes it through an OData end point at the logic layer boundary. OData is a data-access protocol, which you'll learn more about in Chapter 10.

When your LightSwitch client needs to access some data, it doesn't communicate directly with your underlying database or data store. Instead, it accesses data by making an OData call to the data service that's hosted in your logic layer.

There are three places where you can host your application services. The most common place is through an ASP.NET application that's hosted on an IIS (Internet Information Services) server. But you can also host your application services on Windows Azure, or you can host the services on the client workstation in the case of a desktop application.

Note that the word *layer* describes the logical separation of your application, whereas *tier* refers to the physical place where you deploy a layer. Therefore, later chapters in this book use the term 'middle tier' to describe the machine that hosts your logic layer's end point.

Data-retrieval process

The data service exposes operations that return data to the client. By default, these operations include queries that return all records in a table or single records that are filtered by a primary key value. LightSwitch also exposes the queries you create as OData methods.

Figure 1-2 illustrates a typical data operation. When the LightSwitch UI requires data, it calls a query operation on the data service. In this example, the query operation refers to a search query that we've created ourselves to allow users to search for customers by surname. This query allows the client to supply an argument to define the surname to search for.

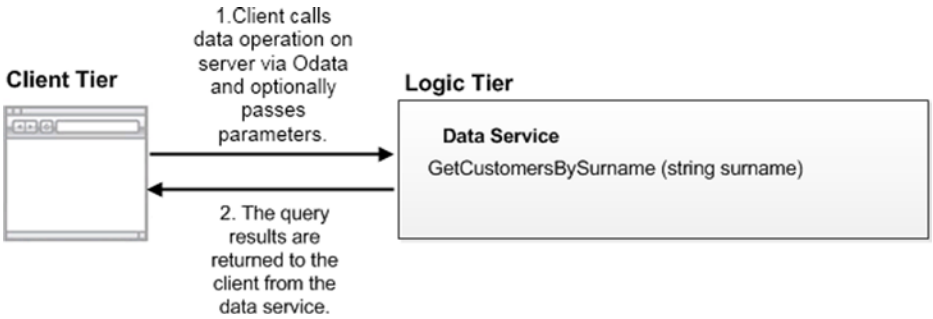

Figure 1-2. *Calling a query that returns customers filtered by surname*

When the data service executes the query, it passes through the *Query Pipeline*. This opens up various phases where you could inject your own custom code. During the pre-processing phase, for example, you can tailor the results of a query by adding extra filter conditions using LINQ (Language Integrated Query) code.

Data-saving process

The data service provides an operation called `SaveChanges`. As the name suggests, this method starts the process that saves any updated data. Let's see how this save process works.

At the very beginning (before `SaveChanges` is called), the client calls a query that returns an initial set of data. The client caches this data using an object called a *Change Set.* This object maintains a record of any records that the user adds, updates, or deletes.

When the user clicks on the save button, the LightSwitch client calls the `SaveChanges` operation and supplies the Change Set. Before the data service commits the changes to the database (or underlying data store such as SharePoint), the processing enters the Save Pipeline.

Just like the Query Pipeline, the Save Pipeline provides places where you intercept the save operation by writing custom code. For example, you can prevent certain users from updating data by including code in the Save Pipeline.

Another feature of LightSwitch is that it prevents users from overwriting changes that have been made by other users. If LightSwitch detects a conflicting data change, it won't allow the save operation to succeed. Instead, it shows what the current and proposed values are and allows the user to choose what to do.

LightSwitch also maintains the consistency of your data by applying all updates inside a transaction. For example, any proposed change to a parent record won't be applied if an update to a child record fails.

Without LightSwitch, you could spend ages writing this type of boring, repetitive boilerplate code. This is an ideal example of how LightSwitch can help you save time.

The Presentation Layer

The presentation layer is the part of LightSwitch that runs on the user's computer or device. This layer actually performs a lot of work. It shows data to the user, allows data entry to happen, and controls all other tasks that relate to human interaction.

Importantly, it also performs business logic such as data validation, keeps track of data changes, and interacts with the Data Services in the logic layer.

You can build your user interface (or client) by using either Silverlight or HTML. In fact, the choice isn't mutually exclusive. You can include both Silverlight and HTML clients in a single LightSwitch application.

Silverlight Client

The advantage of Silverlight is that it provides a rich experience for both developers and end users. You can configure your project so that your application runs as a desktop application or from within a web browser. You can easily switch your application type by clicking a radio button that you'll find in the properties of your LightSwitch project.

An advantage of Silverlight is that it's easier to write code that runs on the client. You can use strongly typed C# or Visual Basic (VB) code and use .NET language features such as LINQ to query your data.

Desktop applications allow you to use COM automation to integrate with the applications that are installed on your end user's machine. This allows you to export the results of data grids to Excel and to automate Office applications such as Word, Excel, and Outlook. LightSwitch hosts your desktop applications using the Silverlight out-of-browser host service (`sllauncher.exe`). It configures your application with elevated permissions to give it access to features such as Export to Excel, COM-based automation, and greater access to the file system.

LightSwitch web applications are hosted by the Silverlight runtime that's installed in your user's web browser. Silverlight browser applications execute inside a sandbox, and access to features such as Export to Excel is prohibited. Silverlight also restricts access to certain parts of the file system.

LightSwitch Shell

You can customize the appearance of a Silverlight application by applying a Shell and a Theme. A Shell controls the location of all of the major UI elements in your application. It also contains the logic that logs users in to your application, and it generates and activates the screens that LightSwitch displays.

Themes define a set of styles that specify the fonts and colors that your application uses. Unlike Shells, Themes apply presentational changes that are much more subtle. Themes are usually designed to work with a specific shell, but it's entirely possible for you to mix and match different Shells and Themes.

Themes and Shells allow you to apply a consistent look and feel throughout your entire application. New developers often struggle to set control attributes (such as fonts and colors) because many of these attributes are designed to be controlled by a Theme, rather than set for each individual control. LightSwitch applies good practice by encouraging you to apply style settings at an application level.

You can set the Shell and Theme for each application through the properties pane of your project. (See Chapter 3.) The two default shells that ship with LightSwitch are called Standard and Cosmopolitan. Figure 1-3 illustrates the same application, but with different Shells applied.

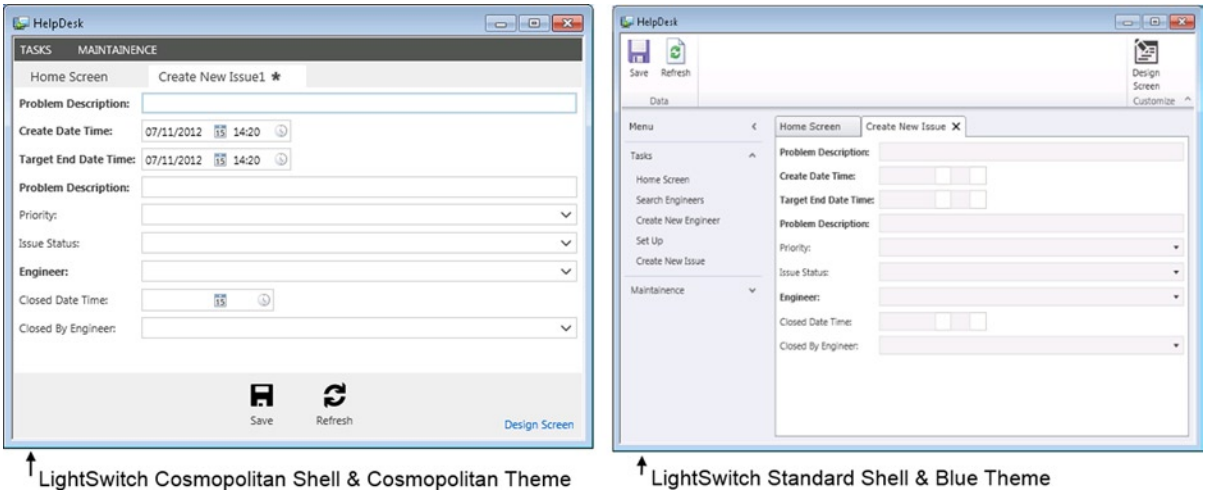

Figure 1-3. *The Cosmopolitan Shell (on the left) and the Standard Shell (on the right)*

If you don't like the Shells or Themes that LightSwitch provides, you can choose to write your own. (This is covered in Chapter 13.)

Understanding Silverlight Screens

A Screen represents a piece of user interface that allows users to view or enter data (just like a Form in an Access application, or a Web Form in an ASP.NET application).

Developers create screens with prebuilt functions by using Templates. For example, there are templates that allow you to create search, editable grid, and data entry screens. You can also create your own screen templates—this is yet another extensibility point that LightSwitch offers.

Users can open multiple screens in a Silverlight LightSwitch application, but there can be only one single active screen at any time.

Each screen contains an object called a *Data Workspace* that is responsible for fetching and managing data. The Data Workspace manages the state of the data in the Change Set that is submitted to the *Data Service* on the logic tier when the user initiates a save.

Because each screen contains its own Data Workspace, the data changes that a user makes in one screen won't be visible in other screens. You can't share data changes between screens because each screen maintains an independent view of the data.

When you're writing Screen code, you'll need some way to reference objects such as data, controls, or screen parameters. LightSwitch exposes these objects through a *Screen Object*, and you'll learn how to use this in Chapter 7.

HTML Client

The *HTML client* is new to LightSwitch 2012. This client is highly recommended for applications that run on mobile or touch-screen devices.

The advantage of using the HTML client is that your application will work on a wider range of devices. In comparison, a Silverlight application that runs in a browser requires the user to install the Silverlight runtime. This isn't always possible, particularly on mobile devices or on locked-down corporate environments.

You can customize your HTML interface by using JavaScript and CSS (Cascading Style Sheets). The advantage is that if you have experience of writing web applications, you can very easily reuse your existing skills.

A disadvantage is that HTML clients are less rich in functionality, and you can't perform COM-related tasks such as automating Microsoft Office. The need to write JavaScript code may also be a disadvantage. Although language choice is largely a matter of taste, many traditional developers find it easier to use a strongly typed .NET language, rather than JavaScript.

Unlike the Silverlight client, the HTML client uses a single document interface. The Silverlight client allows users to open multiple screens and to work on multiple tasks at the same time. In comparison, the HTML client allows users to carry out only one task at a time. Therefore, a key technical difference is that each HTML application contains only one Data Workspace, rather than a separate Data Workspace for each screen.

Introducing the Model-Centric Architecture

LightSwitch's *model-centric* architecture is a brilliant piece of software engineering. It means that your LightSwitch application (data items, screens, queries) is defined and saved in XML files that are named with an LSML extension. When you run your LightSwitch application, the *LightSwitch runtime* processes this XML data and transforms it into a working application.

As a very loose analogy, think of the LightSwitch runtime as a copy of Microsoft Word. Using this analogy, your LightSwitch application would be a Microsoft Word document. In the same way that Word opens Word documents and makes them readable and editable, the LightSwitch runtime opens LightSwitch applications and makes them functional.

The advantage of this architecture is that LightSwitch isn't strongly tied to any specific technology. If Microsoft wants LightSwitch to run on a different platform or device, they can do this by simply writing a new runtime.

Building Your Application

Figure 1-4 shows the parts that make up a LightSwitch application, from design time through to run time. LightSwitch solutions consist of three projects: the Client, Common, and Server projects. These are shown as columns in the diagram. To further explain what happens, let's begin at the bottom.

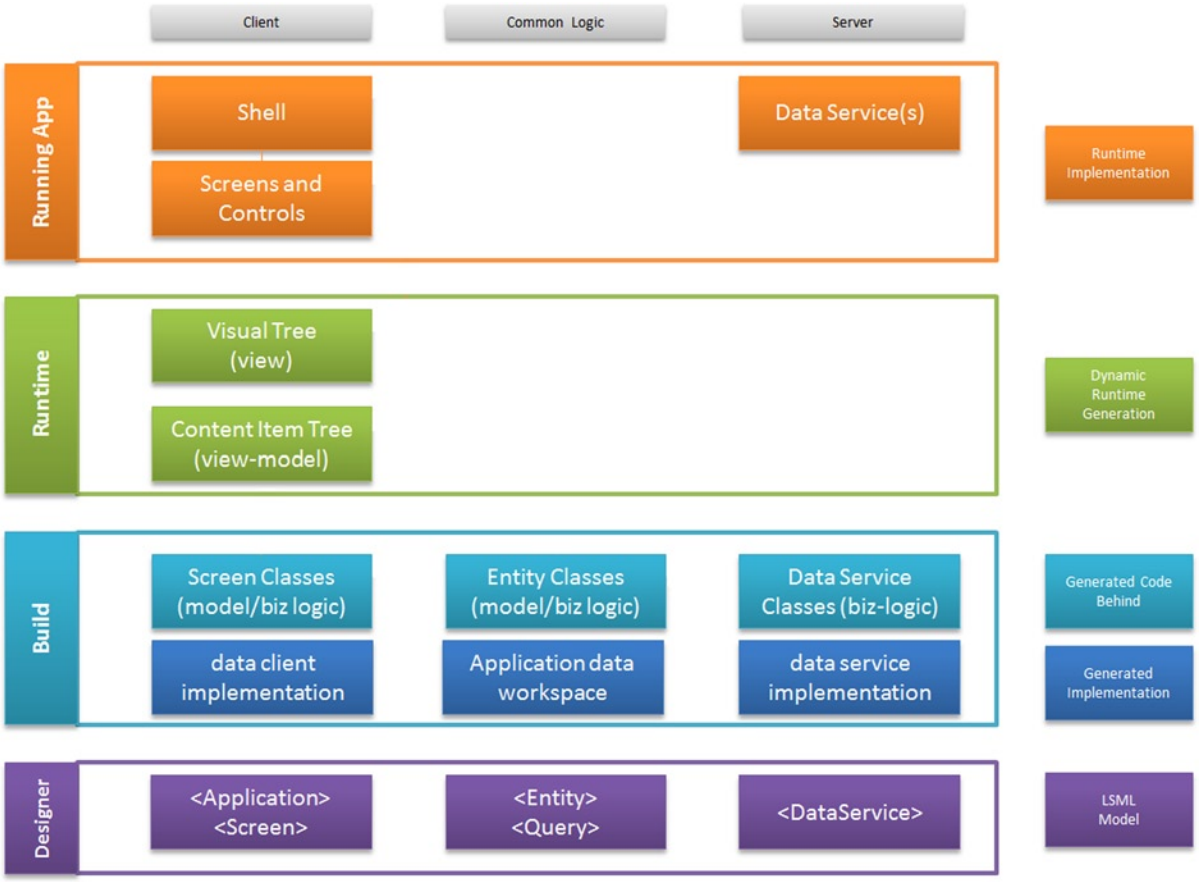

Figure 1-4. *LightSwitch build process*

As part of the model-centric architecture, LightSwitch constructs your applications from building blocks. Think of these as XML representations of the objects that are available in the LightSwitch designer. Let's say that you add a data source to your application. When you do this, LightSwitch expresses your data source as an XML element within the LSML file in your Server project. When you've finished in Visual Studio, your application will be defined entirely in XML, and this is known as the *LSML Model*.

When you build your application, LightSwitch compiles the .NET code that's present in your application. This includes the user code that you've written, in addition to the classes that LightSwitch autogenerates to support your application. The autogenerated code includes the `DataWorkspace` classes and `Screen` object classes. These are core API objects that allow you to access your data through code.

The interesting thing about the LightSwitch clients is that they don't contain any UI that's specific to your application. Instead, the LightSwitch runtime dynamically generates your screens and UI at runtime.

In the case of the Silverlight client, the LightSwitch runtime composes the screens, controls, and shell using the Managed Extensibility Framework (MEF). This dynamic generation prevents you from customizing your Silverlight application by hand-crafting the Extensible Application Markup Language (XAML). This is something that experienced Silverlight developers might be interested in doing. If you want to build your own UI elements using XAML, you can do so by creating your own custom controls (as you'll see in Chapter 11).

In the case of the HTML client, you can customize the HTML that's shown to the user by writing JavaScript that generates custom HTML.

■ **Note** You can create a LightSwitch project in C# or VB.NET. However, you can't change a project from one language to the other after it has been created.

Understanding the Model-View-ViewModel Pattern

LightSwitch applications follow a design pattern called Model-View-ViewModel, or M-V-VM (also known simply as MVVM). This pattern was developed by John Gossman (a technical architect at Microsoft) and is commonly used by Silverlight developers.

MVVM keeps the presentation, logic, and data elements of your application logically distinct. This pattern helps you produce applications that are cleaner, more maintainable, and more testable.

The *Model* part of MVVM refers to the conceptual representation of your data. The entities and queries that you define in your application make up the model.

At runtime, LightSwitch constructs a *screen layout* by interpreting the LSML model. The screen layout consists of a tree of content items, which can represent data items, lists, commands, or parent items. A content item at runtime represents the *View Model* element of MVVM. The View Model controls the client-side business logic (for example, validation) and also manages data access.

In the case of the Silverlight client, the LightSwitch runtime builds the *visual tree*, which contains the actual tree of Silverlight controls that are presented on screen. These Silverlight controls make up the *View* part of MVVM. Views are designed only to display data—nothing more.

In the case of the HTML client, LightSwitch binds HTML controls (the view) to content items that are implemented through JavaScript.

Because views are designed only to display data, it allows you to easily change the control that's bound to a data item. (See Figure 1-5.) This is one of the most striking characteristics of the MVVM pattern.

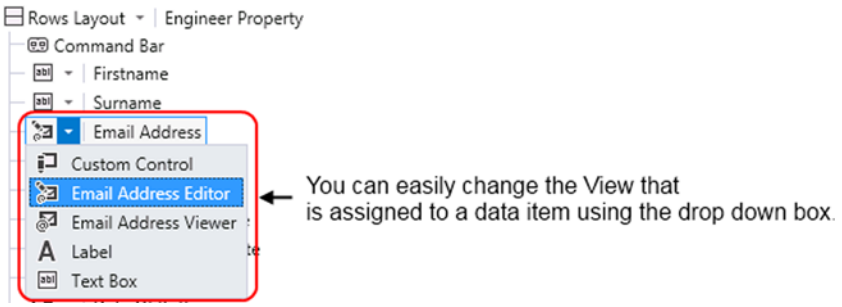

Figure 1-5. *Changing the views that are bound to a data item*

Examining LightSwitch Projects

When you open a project in LightSwitch, you can view your project in one of two ways: Logical View or File View. The default view that LightSwitch presents is the *Logical View*. This basic view organizes your project into various folders such as Data Sources, Entities, and Screens. The Logical View enables you to add and edit items such as screens, queries, and data sources.

File View allows you to see and work on the individual projects that make up your LightSwitch solution. Importantly, it allows you to add references to external dynamic-link libraries (DLLs) and to add your own custom code files.

A button in Solution Explorer allows you to switch between these two views. (See Figure 1-6.) In File View, some files might be hidden by default. Click on the Show All Files button to view all items.

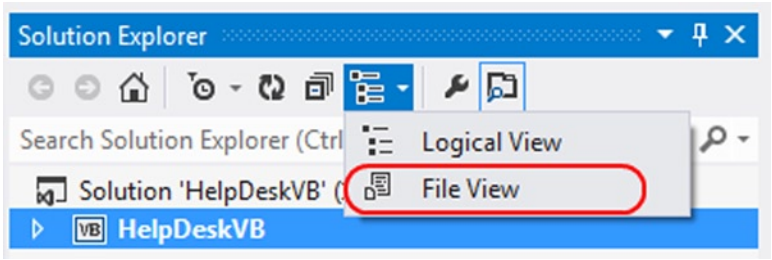

Figure 1-6. *Switching to File View*

EXERCISE 1.1 – EXAMINING LIGHTSWITCH'S SOLUTION STRUCTURE

Start LightSwitch and create a new project. Notice how Solution Explorer shows two folders: Data Sources and Screens. Now switch to File View and examine the underlying projects and files. From Visual Studio's Build menu, select the Build Solution option. After Visual Studio builds your project, use Windows Explorer to view your project files. Notice how LightSwitch generates your build output into a folder called `Bin`. Notice how this folder contains a subfolder called `Data` that contains your design-time Intrinsic database.

After you complete Exercise 1.1, the contents of Solution Explorer will appear as shown in Figure 1-7.

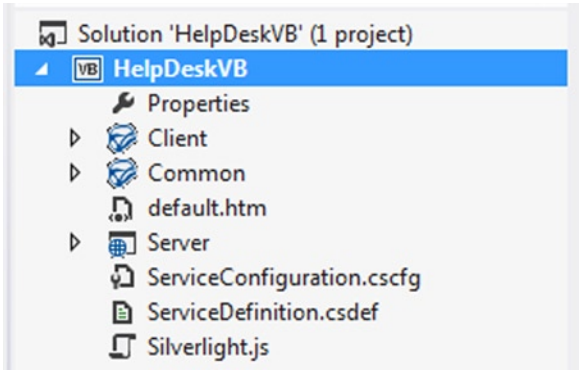

Figure 1-7. *File View of Project*

Notice the `Client`, `Server`, and `Common` projects that make up your solution.

The Client project contains the Silverlight client project, the Server project contains the ASP.NET code that runs on the server, and the Common project contains the logic that runs on both the client and server. Custom validation code is an example of something that belongs in the Common project. This is because LightSwitch executes the same code on both the presentation and logic tiers. In keeping with good practice, LightSwitch validates data on the client to provide instantaneous feedback to the user. It then revalidates on the server to ensure security.

If you create an HTML client project (as discussed in Chapter 8), your project will contain two projects: a Server project, and an HTML Client project. In this scenario, the HTML Client project contains the content that runs on the user's browser. If you add an HTML client to an existing LightSwitch project, LightSwitch no longer uses the code that's in your Common project. Instead, it adds the shared code into the Server project and adds links to the shared code files from the Client project.

■ **Tip** Logical View doesn't allow you to organize your screens, queries, and tables into subfolders. If you're working on a large project, you can keep your project organized by naming the objects that you want to group together with a common prefix.

Reducing Project Sizes

Even the smallest LightSwitch project consumes over 180 MBs in hard disk space. The large project sizes make it difficult to back up, share, or email your LightSwitch projects.

If you want to reduce the amount of space your projects take up on disk, you can safely delete the contents of the `\Client\Bin` and `\Server\Bin` folders. This will reclaim around 130 MBs of space. LightSwitch re-creates these contents when you next compile your application, so there's no need to worry about permanently damaging your project.

Summary

LightSwitch is a rapid development tool that's built with modern technologies, well-known patterns, and best practices. It allows you to easily build data-centric applications that run on the desktop and on the web. LightSwitch applications employ a three-layer architecture and apply the MVVM software design pattern.

LightSwitch relies on a special database called the Intrinsic database. This database stores the tables that you create inside LightSwitch, as well as the login details for your users.

The LightSwitch client application doesn't talk directly with your data sources. Instead, it accesses data by communicating with the logic layer via the OData protocol.

At the logic layer, LightSwitch creates a Data Service for each data source in your application. The data service exposes methods that allow your client to access and update data. When the data service updates or retrieves data for the client, the server-side data operation passes through either the Save or Query Pipelines. These pipelines expose points where you can inject your own custom code.

You can develop LightSwitch user interfaces by using a Silverlight or HTML client. The benefits of a Silverlight client are that it provides a richer experience and can run as a desktop application. The advantage of a desktop application is that it can interact with desktop applications such as Outlook and Word. Silverlight browser applications are unable to do this. The benefits of an HTML application are that it doesn't require the Silverlight runtime and can therefore run on a far wider range of devices. HTML applications are ideal for applications that you want to run on mobile or tablet devices.

LightSwitch uses a Data Workspace object to access data. For each data source, the client caches its working data in an object called a Change Set. In the case of the Silverlight client, each screen has its own Data Workspace; therefore, data changes cannot be shared across screens. The HTML client works differently—in this case, all screens share the same data workspace.

In a Silverlight application, Shells and Themes allow you to re-clothe or skin your application. Changing the Shell allows you to radically change the appearance of your application, whereas changing the Theme allows you to apply more subtle changes, such as changing the font colors and sizes.

LightSwitch solutions consist of three projects. You can see these by switching your solution into File View. This view allows you to add custom classes and references to other .NET assemblies.

LightSwitch applications are defined using XML building blocks, and this XML content is persisted in files with an LSML extension. When you build your application, your LightSwitch clients don't contain any UI that's specific to your application. Instead, LightSwitch autogenerates your application's UI at runtime by interpreting your LSML model.

LightSwitch applies the MVVM pattern throughout your application. A big advantage of this pattern is that it keeps everything logically distinct and allows you to easily change the controls that are bound to data items.

CHAPTER 2

Setting Up Your Data

LightSwitch's greatest strength is how easy it makes working with data. So to get the most out of the product, it's vital that you learn how to do that!

This chapter teaches you how to

- Design tables and attach external data sources to your application
- Define relationships between tables
- Create computed properties and apply business types

This chapter traces the genesis of an application that manages help desk issues. It provides a real-life demonstration of a typical application that you'll find in everyday business, and includes sufficient complexity to demonstrate most of the features in LightSwitch.

In this chapter, you'll find out how to create tables that store help desk issues and engineer details. You'll learn how to associate engineers with multiple issues and how to define a manager/engineer hierarchy by defining relationships. To help users identify engineers in lists of data, you'll learn how to create a computed property and define a summary property. The computed property summarizes the first name and surname of each engineer, and provides a friendly record descriptor for each engineer record.

Choosing Where to Store Your Data

There are two approaches for storing data in LightSwitch. You can create your own tables in the Intrinsic database by using the built-in table designer, or you can attach to an external data source. Of course, these two approaches are not mutually exclusive. You can create your own tables and also attach as many external data sources as you want.

When you build tables using the built-in table designer, LightSwitch persists any data that you add at design time. So if you add some data to a table during a debug session, your data will still be there during the next debug session. For this to work, LightSwitch creates a `LocalDB` development database in the location `\Bin\Data\Temp\ApplicationDatabase.mdf`.

The advantage of creating tables in the Intrinsic database is that your project is self-contained. If you share your LightSwitch project with other users, they'll be able to run your project without having to reattach the external data.

The difficulty arises when you deploy your application. The deployment wizard doesn't allow you to deploy your development data into your live environment. This can be frustrating if you've spent a lot of time entering data during the design process. So if design time data is important to you, you should consider building your tables externally in a Microsoft SQL Server database rather than building it internally within LightSwitch.

What Are Entities and Properties?

Database developers often use the terms *tables* and *rows*. However, LightSwitch regularly refers to *entities* and *properties* instead. An entity represents the data from a single row in a database table, whereas a property is analogous to a field or column from a database table.

LightSwitch uses Microsoft's Entity Framework internally to provide object relational mapping. Entity and Property are terms that the Entity Framework uses, and are more appropriate given that LightSwitch can connect to nonrelational data sources. For example, if you connect to a SharePoint data source, list items map to LightSwitch entities, and list columns are exposed as LightSwitch properties.

In this book, I use the words *tables* and *fields* interchangeably because it's often clearer to use the database terms. For example, a property in the screen designer can mean a local instance of an entity or something that you can find in Visual Studio's property sheet. And once you start talking about the properties of a property, things can quickly become quite confusing.

Creating Tables (Entities)

Let's start by creating a new table and adding some fields. We'll create a table called `Engineers` that stores details about the engineers.

In Solution Explorer, right-click the Data Sources folder and choose the Add Table option. Once you've added your table, you can modify the properties using the Properties sheet. (See Figure 2-1.)

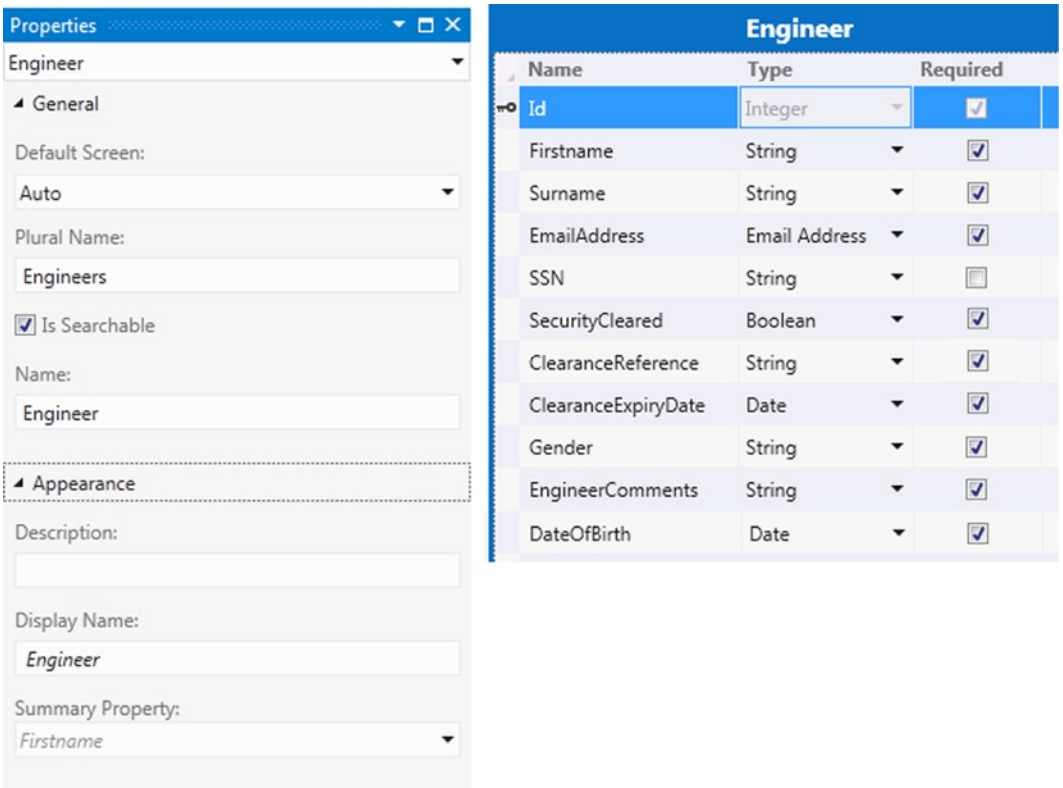

Figure 2-1. *Creating a table and editing its properties*

Some of the table properties that you can set include

- **Name:** The name uniquely identifies your table. The name must begin with an alphabetic character and can contain only alphanumeric characters and the underscore character. Other special characters, such as spaces or the period character, are not permitted.
- **Display name:** This is the friendly name that describes your table—it can contain spaces.
- **Description:** The description field provides a long description of the table.
- **Plural Name:** LightSwitch uses the value that you enter here to name the collections of entities that you add to a screen. You would also use the plural name to refer to collections of entities when you're writing code.
- **Summary Property:** this allows you to specify the property that identifies a data row to a user. You can add a control to your screen called a Summary Control that exposes the value of this property to the user.
- **Default Screen:** The Summary Control renders itself as a link. When a user clicks on this link, it opens the default screen that's specified here. There is an option to select Auto from the drop-down list. If you select Auto, LightSwitch displays an autogenerated screen.
- **Is Searchable:** Checking this defines that the text properties in the table will be searchable.

Creating Fields (Properties)

After creating the `Engineer` table, add the fields (or properties) that are shown in Figure 2-1. When you add a field to a table, you'll need to specify the type of data that'll be stored in the field. Let's take a look at some of the data types that you can use.

■ **Caution** Try not to name your fields using words that are reserved keywords in SQL Server or the Entity Framework. Prefixing field names with the word *Entity* (for example, `EntityKey`) caused unexpected errors in LightSwitch 2011. Although Microsoft has fixed this particular bug, it's safer never to name your fields after reserved keywords because you never know what unexpected errors might occur.

Storing String Data

The string data type includes support for international character types (for example, Arabic, Chinese, and Japanese). For maximum performance, it's a good idea to set the maximum field size to a value that's appropriate for the data that you want to store.

If you want to store unlimited length or multiline text, clear the contents of the maximum field text box and leave it blank, as shown in Figure 2-2.

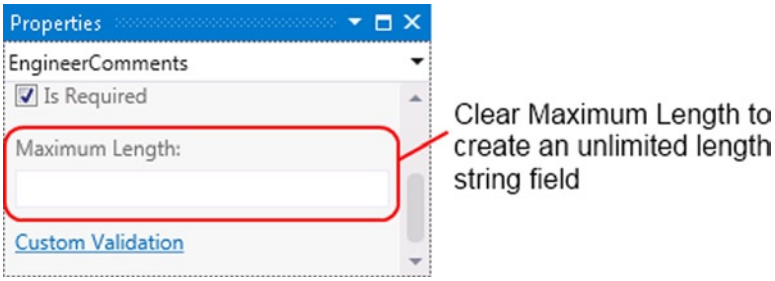

Figure 2-2. *Setting a string field to store text with unlimited length*

Storing Numbers (Double and Decimal Types)

If you want to store numbers with decimal places, LightSwitch provides a choice of either double or decimal data types.

The practical difference between the two types is that doubles can store a wide range of numbers in a smaller amount of memory. However, doubles are less precise and are subject to rounding errors when calculations are performed against them.

Decimals don't suffer from such rounding errors but take up more space and are slower to compute. Sums of money should always be based on the decimal data type.

Other important attributes that relate to decimals are precision and scale. *Precision* defines the total number of digits in a number. *Scale* defines the number of digits after the decimal place. Figure 2-3 illustrates precision and scale using the example number 123456.789.

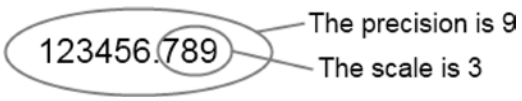

Figure 2-3. *Precision and scale*

When you create a new decimal field, LightSwitch defaults the precision and scale values to 18 and 2, respectively. In keeping with good practice, you should shorten these values if you don't require that level of accuracy.

Formatting Numeric Fields

This is a new feature in LightSwitch 2012. You can specify a display format for each numeric property that you define in LightSwitch. This means that LightSwitch formats the number on the user's screen using the format that you've specified.

Figure 2-4 shows a field that stores a feedback rating. A format string of `N2` means that the number is shown to 2 decimal places.

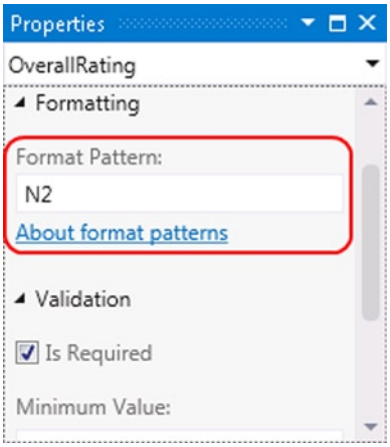

Figure 2-4. *Setting the format*

.NET format strings begin with a Format Specifier, followed by a number that indicates the desired number of decimal places. Table 2-1 shows the Format Specifiers that you can use.

Table 2-1. *.NET format specifiers*

Format Specifier	Description
C or c	Currency
D or d	Decimal
E or e	Scientific (exponential)
F or f	Fixed-point
G or g	General
N or n	Number
P or p	Percent
R or r	Round-trip
X or x	Hexadecimal

■ **Note** LightSwitch 2011 didn't allow you to format numeric fields with .NET format strings. In March 2011, I submitted this idea on Microsoft Connect, a web site that allows you to provide feedback on how to improve Microsoft products: https://connect.microsoft.com/VisualStudio/feedback/details/654220/lightswitch-allow-data-to-be-formatted.

I'm very pleased that Microsoft has added this useful feature. This goes to show that if you have any ideas on how to improve LightSwitch, it's worth recording them through Connect or the Uservoice web site (http://visualstudio.uservoice.com/).

Storing Images

The Image Type allows you to store images. LightSwitch includes Image Editor and Image Viewer controls that allows users to upload and view images.

Note that these controls support only images in JPG and PNG format. If you want to upload and view image files in other formats, you'll need to write or purchase a third-party custom control.

Storing Binary Data

You can use the binary data type to store binary large objects such as documents, videos, or other file types. Chapters 7 and 8 show you how to design a screen that allows users to upload and download files.

Ensuring Unique Values

Each field contains an Include In Unique Index check box. Selecting this check box adds the field into a combination index.

It's not possible to create individual unique fields through the designer. If you want to do this, you can write validation to enforce uniqueness at a field level. Chapter 5 contains sample code that shows you how to do this.

Alternatively, you could create unique indexes on your SQL Server table if you're using an attached SQL database.

Changing Data Types

If you're working on tables in your Intrinsic database and make some major table changes, LightSwitch can discard the data in your tables. At worst, it can even destroy and re-create your entire development database. However, it generally warns you before any data loss occurs. (See Figure 2-5.)

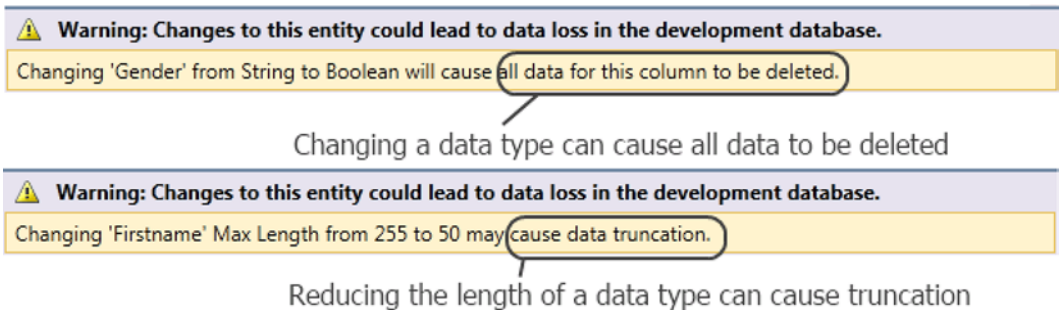

Figure 2-5. *Warnings that appear before data loss occurs*

Using LightSwitch Business Types

You'll find some special data types that you won't find in other database management systems. These include Email, Money, Phone Number, Web Address, and Percent. These data types are called *business types*. They're specially designed to store specialized data and provide built-in validation and data entry controls. These business types include properties that control how LightSwitch displays your data on screen, as well as data entry and validation characteristics. Let's take a closer look at some of these business types.

Storing Email Addresses

As its name suggests, the Email business type provides storage for email addresses.

When you add an Email field, you'll find two properties that you can set (as shown in Figure 2-6):

- **Default Email Domain:** If the user leaves out the email domain, LightSwitch appends the default email domain that you specify to the end of the email address. This setting is ideal for internal systems that are used by a single company.
- **Require Email Domain:** If this is checked, the user must enter an email domain when entering an email address.

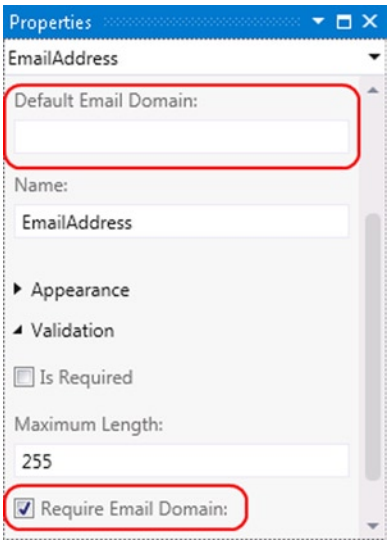

Figure 2-6. *The two Email business type properties*

Storing Money Values

When you add a Money field to a table, the additional properties that you can set are shown, as you can see in Figure 2-7. These are

- **Currency Code:** Use this field to specify the locale for the currency. For example, if you want to specify United States Dollars, specify USD. Appendix A shows a list of valid codes that you can use.
- **Is Formatted:** If you check this, LightSwitch applies formatting when you use the currency control on a screen to display your money value. The formatting that the control applies includes the currency symbol, the grouping separator, and decimal places.

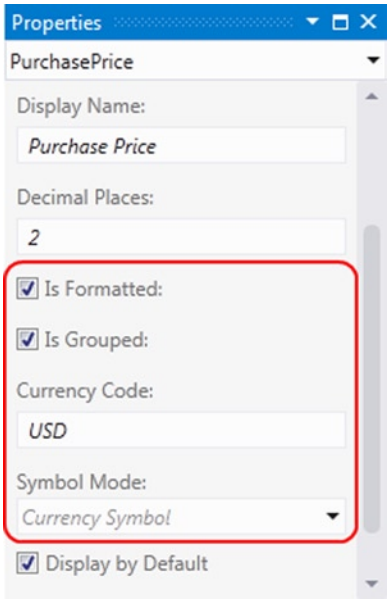

Figure 2-7. *Properties that you can set on a Money field*

The following options apply only if you check the Is Formatted check box:

- **Is Grouped:** If checked, LightSwitch shows digit-grouping separators. For example, it displays 1,234,567.89 rather than 1234567.89.
- **Symbol Mode:** The symbol mode dropdown allows you to select Currency Symbol, ISO Currency Symbol, or No Currency Symbol. Here are some examples of how LightSwitch would format a money value using the available symbol modes:
  - Currency Symbol: $123.45
  - ISO Currency Symbol: 123.45 USD
  - No Currency Symbol: 123.45
- **Decimal Places:** This defines the number of decimal places that LightSwitch shows when it formats a money value.

Storing Phone Numbers

The Phone Number business type is designed to store phone numbers, and it validates data by making sure that users can enter only phone numbers that match a list of predefined formats.

You can define formats by using the dialog that's shown in Figure 2-8. The symbols that you can use to define a format are

- C – Country Code
- A – Area or City Code
- N – Local number

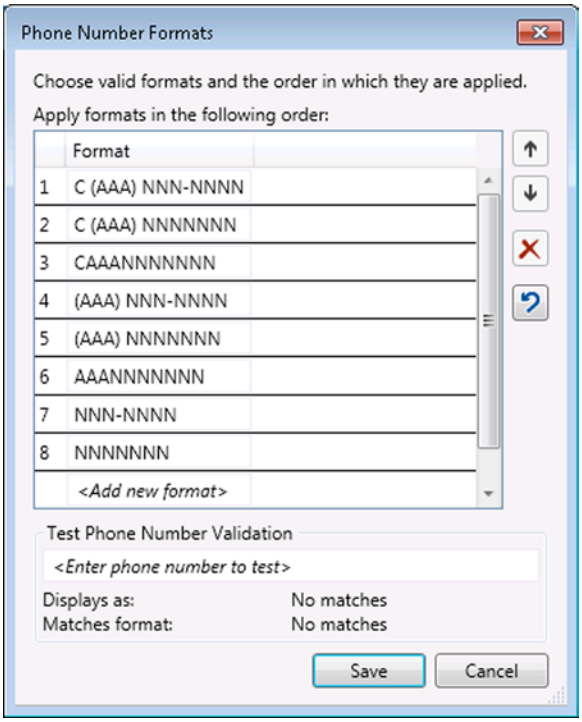

Figure 2-8. *Phone Number Formats dialog*

Other additional symbols that you can use include the following: +, –, (,), .

When a user enters a phone number through the Phone Number control, the control attempts to validate the phone number against the first format in the list. If the digits match the format, LightSwitch displays the phone number using that format. Otherwise, the control attempts to validate the phone number against all remaining formats in the list until a match is found. If it doesn't find a match, LightSwitch prevents the record from being saved.

When LightSwitch saves the phone number in the database, it does so without any formatting. If you want to create reports or use the phone number data outside of LightSwitch, you'll need to write your own procedure to format the data.

Unfortunately, it isn't possible to specify additional formats on a global or application basis. They must be specified each time a Phone Number field is used in a table.

Storing Web Addresses and Percentage Values

The Web Address business type is new to LightSwitch 2012. It allows you to store Web addresses, and it allows users to add and edit data using the Web Address Editor and Web Address Viewer controls.

Also new to LightSwitch 2012 is the Percent business type. This Business Type allows you to store percentage values. Just like the other business types, LightSwitch provides a Percent Editor and Percent Viewer control for data entry.

Examining What Happens in SQL Server

When you create a table in the table designer, LightSwitch creates the actual table in your Intrinsic SQL Server database. If you create a string property of 255 characters, LightSwitch creates a 255 length `nvarchar` column (a data type that stores variable length unicode data) in the underlying SQL Server table.

LightSwitch provides an API that allows you to access any property that's visible in the table designer through code. When you're writing code, LightSwitch exposes table properties using .NET data types. The mapping between LightSwitch, .NET, and SQL Server data types is shown in Table 2-2.

Table 2-2. *Mappings between LightSwitch, .NET, and SQL data types*

LightSwitch Type	VB.NET Type	C# Type	SQL Type
Binary	Byte()	byte[]	varbinary(max)
Boolean	Boolean	bool	Bit
Date	DateTime	DateTime	Datetime
DateTime	DateTime	DateTime	Datetime
Decimal	Decimal	decimal	Decimal
Double	Double	double	Float
Email Address	String	string	Nvarchar
Image	Byte()	byte[]	varbinary(max)
Short Integer	Short	short	Smallint
Integer	Integer	int	Int
Long Integer	Long	long	Bigint
Money	Decimal	decimal	decimal(18,2)
Phone Number	String	string	Nvarchar
Percent	Decimal	decimal	decimal(18,2)
String	String	string	Nvarchar
Web Address	String	string	Nvarchar

In Table 2-2, notice the business type mappings. LightSwitch uses SQL `nvarchar` and numeric types as the underlying storage type for business types.

■ **Tip** Notice how LightSwitch `Date` and `DateTime` fields map to the SQL DateTime data type. The minimum date value that you can store is 01-Jan-1753. If you need to store earlier dates (for example, if you're writing genealogy software), create your tables in an external SQL database and create columns using the `DateTime2` datatype (a data type that offers more precision and a greater range of date values).

Creating Choice Lists

Choice Lists allow you to use the Auto Complete Box or Modal Window Picker control to restrict the data values that users can enter. You can specify choice lists on fields from external data sources, as well as on tables that you create yourself in the Intrinsic database.

A choice list is a set of name value pairs, and you can configure these for most data types, including Boolean and numeric types. For example, you can create a choice list for a Boolean field with the choice values of Yes and No. The data types that don't support choice lists are the image and binary types.

To create a choice list, click the Choice List link on the properties sheet for your field and enter a list of Value and Display Name pairs. When the choices are shown on screen, they'll appear in the order in which they're entered. You can re-order the choices by right-clicking the item and selecting the Move Up or Move Down option, as illustrated in Figure 2-9.

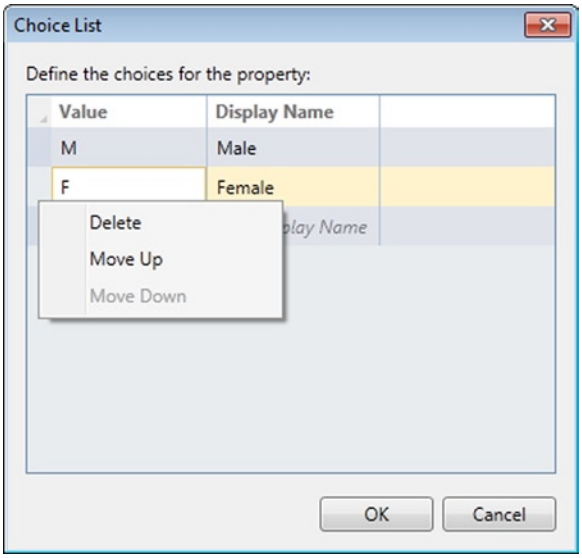

Figure 2-9. *Reorder items using the right-click context menu*

LightSwitch saves your choice list settings in the following file: `Common\My Project\Common.lsml`. It's possible to maintain your choice list entries by editing this file manually in Notepad. This could be useful if you want to speed up data entry by adding multiple choices using copy and paste, rather than adding them one by one through the Choice List dialog.

■ **Caution** Always back up and take care before you manually modify an LSML file. You can irreparably damage your LightSwitch solution if you make a mistake. Without a recent backup, you can end up having to re-create your entire LightSwitch solution and redo all of your work.

Choice Lists vs. Related Tables

If you want your users to enter data by choosing from a list of available choices, the choice values can either come from a choice list or a related table.

A choice list is ideal for data items that are relatively static. The disadvantage of using a choice list is that adding or deleting items requires you to recompile and redeploy your application, which can be cumbersome. Table 2-3 summarizes the pros and cons of choice lists and related tables.