# Contents at a Glance

## Part I Building Applications with LightSwitch

1. Introducing Visual Studio LightSwitch ................................................ 1  
2. Exploring the IDE .......................................................................... 23  
3. Building Data-Centric Applications ................................................... 43  
4. Building More-Complex Applications  
   with Relationships and Details Screens .............................................. 95  

## Part II Manipulating Data

5. Customizing Data Validation .......................................................... 149  
6. Querying, Filtering, and Sorting Data ............................................... 177  
7. Customizing Applications with Buttons,  
   COM Automation, and Extensions .................................................. 215  
8. Aggregating Data from Different Data Sources ................................... 235  

## Part III Securing and Deploying Applications

10. Deploying LightSwitch Applications ................................................ 339  

## Part IV Advanced LightSwitch

11. Handling Events in Code ............................................................... 409  
12. Dissecting a LightSwitch Application ............................................... 443  
13. Advanced LightSwitch with Visual Studio 2010 ................................. 479  
14. Debugging LightSwitch Applications ............................................... 517  

## Part V Extensibility

15. Customizing the IDE ..................................................................... 543  
16. Customizing Applications with Custom Controls .............................. 561  
17. Implementing Printing and Reporting .............................................. 581  
18. LightSwitch Extensibility: Themes, Shells, Controls, and Screens .... 619  
19. LightSwitch Extensibility: Data and Extension Deployment ................. 693  

## Appendixes

A. Installing and Configuring Visual Studio LightSwitch ......................... 759  
B. Useful Resources ........................................................................... 765  

Index ................................................................................................. 771
# Table of Contents

## Part I  Building Applications with LightSwitch

1  **Introducing Visual Studio LightSwitch**  
Who Uses LightSwitch? ................................................................. 3  
A Short History of Microsoft Business Tools ............................. 3  
   Microsoft Access with Visual Basic for Applications ............ 4  
   Microsoft Visual Basic 6 ......................................................... 6  
   Microsoft Visual FoxPro ......................................................... 9  
   Microsoft Visual Studio .NET (2002 to 2010) ....................... 10  
About Visual Studio LightSwitch ............................................. 13  
   Technologies Used Behind the Scenes ................................. 15  
   Available Editions ................................................................ 18  
   Companion Source Code ..................................................... 18  
Setting Up the Development Machine ....................................... 19  
   Operating System .................................................................. 19  
   Development Environment .................................................. 19  
   Server Components ................................................................ 19  
   Database Tools ..................................................................... 20  
   Controls and Toolkits ........................................................... 21  
What You Need to Know About Programming ......................... 21  
Summary .................................................................................. 22

2  **Exploring the IDE**  
Getting Started with Visual Studio LightSwitch ......................... 23  
   Introducing the Start Page .................................................... 25  
   Creating New Projects and Exploring Solutions ...................... 26  
   The LightSwitch Designer ..................................................... 29  
   The Entity Designer .............................................................. 30  
   The Screen Designer ............................................................ 30  
   The Properties Window ....................................................... 32  
   The Query Designer ............................................................. 33  
   The Application Designer .................................................... 34  
   The Code Editor .................................................................. 35  
   Building, Running, and Debugging Applications ................... 35  
   Managing and Arranging Windows ...................................... 37  
Getting Help ............................................................................. 38  
Visual Studio 2010 with LightSwitch ....................................... 39  
Summary .................................................................................. 41
3 Building Data-Centric Applications

Creating a New Application ......................................................... 43
Creating a New Data Source ............................................................ 45
    Adding Entity Properties ......................................................... 46
    Building a Complete Entity .................................................... 50
    Data Storage ........................................................................... 52
The User Interface: Implementing Screens .......................................... 52
    Controls Overview .................................................................. 53
    Creating a Data Entry Screen .................................................. 55
    Creating a Search Screen ....................................................... 58
Testing the Application on the Development Machine ......................... 60
    Starting the Application as a Desktop Client ............................ 61
    Adding and Editing Data ........................................................ 63
    Displaying and Searching Data ............................................... 68
    Exporting Data to Microsoft Excel ......................................... 77
    Basic Screen Customizations ................................................... 78
    Running the Application as a 3-Tier Desktop Client .................. 83
    Running the Application in the Web Browser ............................ 84
Input-Data Validation ..................................................................... 85
    Required Fields Validation ..................................................... 85
    String-Length Validation ....................................................... 86
    Date Validation .................................................................... 87
    Number Validation .................................................................. 90
Default Validation of Business Types ................................................. 91
    Validating Email Addresses.................................................... 91
    Validating Phone Numbers .................................................... 93
    Validating Images ................................................................ 93
    Validating Money ................................................................... 93
Summary ....................................................................................... 93

4 Building More-Complex Applications with Relationships and Details Screens

Creating a New LightSwitch Project .................................................. 96
Designing Complex Data Sources ...................................................... 97
    Entities That Define Choice Lists ......................................... 100
    Working with the Money Data Type ........................................ 103
    Adding Relationships ............................................................ 105
    Using Computed Properties ................................................ 112
More-Complex, Business-Oriented User Interfaces .............................. 115
    Creating Data Entry Screens ................................................... 115
    Creating Search Screens ....................................................... 116
    Running the Application and Entering Data ............................ 116
    Editing Data with Editable Grids ............................................. 118
8 **Aggregating Data from Different Data Sources** 235

Connecting to SQL Server Databases ................................................ 236
  Installing the Sample Northwind Database .................................. 237
  Creating New Applications on Existing SQL Server Databases ....... 239
  Aggregating Data from Existing Databases into LightSwitch Applications ................................................ 249

Database in the Cloud: Connecting to SQL Azure .............................. 256
  Establishing Connections to SQL Azure .................................... 259
  Creating a SQL Azure Database ............................................... 262
  Connecting LightSwitch Applications to SQL Azure .................... 267

Collaboration: Working with Lists from SharePoint 2010 .................... 273
  Running the Default Website .................................................. 274
  Configuring a Vacation Plan Calendar on SharePoint 2010 .......... 275
  Extending LightSwitch Applications with SharePoint Data ........... 280

Summary .................................................................................... 290

Part III **Securing and Deploying Applications**

9 **Implementing Authentication and Authorization** 291

Understanding Authentication ....................................................... 292
Implementing Windows Authentication ........................................... 294
  Setting Up the Development Environment .................................. 296
  Authorization: Settings Permissions ................................. 297
  Writing the Permission Logic ................................................ 298
  Permission Logic on Entities ................................................... 299
  Permission Logic on the User Interface ............................... 307
  Debugging the Application ..................................................... 312
  Creating User Roles and Administering the Application .............. 312
Implementing Forms Authentication ............................................... 329
  Publishing Applications with Forms Authentication ............... 330
  Testing the Application with Different Credentials ............... 334
Permission Elevation on Server Code ............................................... 336
Summary .................................................................................... 338

10 **Deploying LightSwitch Applications** 339

Deployment Fundamentals ............................................................ 339
  Understanding 2-Tier and 3-Tier Applications ......................... 341
Preparing the Application for Deployment .................................... 344
  Specifying the Application Name and Logo ......................... 345
Styling the Application .......................................................... 348
Runtime Settings .................................................................. 349
Localizing an Application ................................................................. 350
## Deploying 2-Tier Applications
- Client Configuration .......................................................... 353
- Application Server Configuration ............................................ 353
- Publish Output ..................................................................... 353
- Database Connections ........................................................... 356
- Prerequisites ........................................................................ 357
- Other Connections ............................................................... 360
- Specifying a Certificate .......................................................... 363
- Publish Summary ................................................................. 366
- Publishing and Deploying the Application .............................. 368

## Deploying 3-Tier Applications
- Configuring the Target Web Server .......................................... 374
- Publishing the Application ..................................................... 377
- Remotely Publishing to a Web Server ....................................... 379
- Creating and Importing Packages into IIS ............................... 385

## Deploying to Windows Azure
- Preparing Applications to Deployment ..................................... 393

## Summary

## Part IV Advanced LightSwitch

### 11 Handling Events in Code
- Working with Entities in Code .................................................. 409
  - Understanding Data Objects ................................................... 410
  - Handling Data Events in Code ................................................ 413
- Handling Custom Query Events in Code .................................... 429
- Handling Screen Events in Code ............................................. 430
  - Button Methods .................................................................. 431
  - General Methods .................................................................. 434
  - Collection Methods ........................................................... 437
- Launching Screens Programatically ........................................ 438
  - Opening Screens Without Passing Parameters ..................... 440
  - Opening Screens by Passing Parameters ............................... 440

### 12 Dissecting a LightSwitch Application
- Applications Architecture and Tiers ......................................... 444
  - Architecture Overview ......................................................... 444
  - Understanding the Data Access and Storage Tiers ................. 445
  - Understanding the Logic Tier ................................................ 452
  - Understanding the Presentation Tier ..................................... 462
- Dissecting LightSwitch Projects .................................................. 469
  - Overview of LightSwitch Solutions ........................................ 470
  - Server-Side Projects ............................................................ 471
13 Advanced LightSwitch with Visual Studio 2010

Managing the Application Life Cycle with TFS 2010
  Connecting to Team Foundation Server
  Creating New Team Projects
  Submitting Projects to Source Control
  Creating and Assigning Work Items
  Retrieving Specific Project Versions with Version Control
  Automating Builds

Code Metrics
  Unit Testing Your Helper Code
    Creating a Test Project
    Creating Unit Tests
  Analyzing the Execution Flow with IntelliTrace
    IntelliTrace Options
    IntelliTrace in Action with LightSwitch
    IntelliTrace Log Files

Summary

14 Debugging LightSwitch Applications

Debugging Applications
  The Error List Tool Window
  Breakpoints and Data Tips
  Debugging in Steps
  About Runtime Errors
  The Edit and Continue Feature

Advanced Debugging Instrumentation
  Breakpoints and Trace Points Unleashed
  Showing a Variables Value in the Locals Window
  Evaluating Expressions in the Command Window
  Understanding the Method Calls Flow in the Call Stack Window
  Watch Windows
  Analyzing Threads with the Threads Window

Understanding Debugger Visualizers

Debugging in Code
  The Debug Class
  Using Debugger Attributes in Code

Summary
### Part V Extensibility

#### 15 Customizing the IDE

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customizing Visual Studio LightSwitch</td>
<td>543</td>
</tr>
<tr>
<td>Customizing Commands and Toolbars</td>
<td>544</td>
</tr>
<tr>
<td>Customizing an Existing Toolbar</td>
<td>544</td>
</tr>
<tr>
<td>Creating a New Custom Toolbar</td>
<td>544</td>
</tr>
<tr>
<td>Managing User Settings</td>
<td>546</td>
</tr>
<tr>
<td>Exporting Settings</td>
<td>547</td>
</tr>
<tr>
<td>Importing Settings</td>
<td>548</td>
</tr>
<tr>
<td>Using, Creating, and Managing Reusable Code Snippets</td>
<td>550</td>
</tr>
<tr>
<td>Using Code Snippets</td>
<td>551</td>
</tr>
<tr>
<td>The Code Snippet Manager</td>
<td>553</td>
</tr>
<tr>
<td>Creating and Using Custom Code Snippets</td>
<td>554</td>
</tr>
<tr>
<td>Summary</td>
<td>559</td>
</tr>
</tbody>
</table>

#### 16 Customizing Applications with Custom Controls

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Building Custom Controls</td>
<td>561</td>
</tr>
<tr>
<td>Creating Controls with Visual Studio 2010</td>
<td>562</td>
</tr>
<tr>
<td>Implementing and Aggregating Chart Controls</td>
<td>563</td>
</tr>
<tr>
<td>Binding the User Control to Data and the Screen</td>
<td>565</td>
</tr>
<tr>
<td>Integrating Bing Maps</td>
<td>569</td>
</tr>
<tr>
<td>Claiming Your Bing Maps Developer Keys</td>
<td>573</td>
</tr>
<tr>
<td>Making Bing Maps Available to LightSwitch</td>
<td>574</td>
</tr>
<tr>
<td>Updating Entities to Support Bing Maps</td>
<td>575</td>
</tr>
<tr>
<td>Adding Bing Maps to Screens</td>
<td>576</td>
</tr>
<tr>
<td>Summary</td>
<td>580</td>
</tr>
</tbody>
</table>

#### 17 Implementing Printing and Reporting

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Using the Office Integration Extension</td>
<td>581</td>
</tr>
<tr>
<td>Overview of the Class Library</td>
<td>582</td>
</tr>
<tr>
<td>Creating Emails and Appointments with Microsoft Outlook</td>
<td>583</td>
</tr>
<tr>
<td>Exporting to Microsoft Word and to PDF</td>
<td>585</td>
</tr>
<tr>
<td>Importing and Exporting Data with Microsoft Excel 2010</td>
<td>586</td>
</tr>
<tr>
<td>Using SQL Server Reporting Services</td>
<td>593</td>
</tr>
<tr>
<td>Reporting Extensions from Microsoft Partners</td>
<td>597</td>
</tr>
<tr>
<td>XtraReports from Developer Express</td>
<td>599</td>
</tr>
<tr>
<td>NetAdvantage for LightSwitch from Infragistics</td>
<td>600</td>
</tr>
<tr>
<td>OLAP Extension from Component One</td>
<td>605</td>
</tr>
<tr>
<td>Telerik Reporting for Silverlight</td>
<td>606</td>
</tr>
</tbody>
</table>
18 LightSwitch Extensibility: Themes, Shells, Controls, and Screens 619
Understanding the Extensibility Model 620
Creating Themes 621
Creating a New Extensibility Project 621
Setting Extension Properties 624
Adding a Theme to the Extensibility Project 625
Editing Themes with Visual Studio 2010 628
Editing Themes with Expression Blend 4 630
Making the Green Theme 631
Testing the Custom Theme 633
Creating Custom Shells 636
Creating Extensibility Projects for Custom Shells 638
Editing the Official Sample from Microsoft 639
Sharing Custom Controls 656
Available Control Types 657
Creating Control Extensions 657
Creating Screen Templates 670
Designing Custom Screen Templates 672
Implementing the IScreenTemplate Properties 673
Generating the Screen Content Tree 675
Adding Code to the Screen 678
Testing the Screen Template 685
Screen Templates Tips and Tricks 686
Summary 691

19 LightSwitch Extensibility: Data and Extension Deployment 693
Creating Business Types 693
Implementing a New Business Type 693
Implementing the Data Type Definition 694
Implementing Data Validation 696
Designing Controls 700
Testing the Extension 703
Creating and Using Custom Data Sources with WCF RIA Services 705
The .NET Framework for WCF RIA Services 709
Creating WCF RIA Services to Work with XML Data 711
Calling WCF RIA Services from LightSwitch Applications 720
Calling Stored Procedures Through WCF RIA Services 726
Appendixes

A Installing and Configuring Visual Studio LightSwitch

Installing Visual Studio LightSwitch ................................................ 759
Managing the Offline Documentation ............................................. 762

B Useful Resources

MSDN Resources .......................................................................... 765
  LightSwitch Developer Center .................................................... 765
  LightSwitch How-Do-I Videos ...................................................... 766
  LightSwitch Blogs ...................................................................... 766
  LightSwitch Forums .................................................................... 766
  Training Kit and Starter Kits ...................................................... 766
  Extensibility Center .................................................................. 767
  The Visual Studio Gallery ......................................................... 767
  Code Samples ........................................................................ 767
  Social Networks ....................................................................... 767
Community Resources ................................................................... 767
  Visual Studio LightSwitch Help Website .................................... 767
  LightSwitch Video Training on MyVBProf.com ......................... 768
  The CodeProject ..................................................................... 768
  StackOverflow ........................................................................ 768
  Italian LightSwitch Tips & Tricks Community ............................ 768
  Other LightSwitch Blogs and Websites ...................................... 768
Third-Party Extensions .............................................................. 769
  Document Toolkit for LightSwitch ............................................ 769
  RSSBus Data Providers ............................................................ 769
  Infragistics NetAdvantage Light Edition .................................... 770

Index ........................................................................................................ 771
About the Author

Alessandro Del Sole has been a Microsoft Most Valuable Professional (MVP) for Visual Basic since 2008. He is well known throughout the global Visual Basic community worldwide. He is a community leader on the Italian Visual Basic Tips & Tricks website (more than 44,100 developers) and on the LightSwitch Tips & Tricks website. He is a frequent contributor to the Visual Basic Developer Center on MSDN and the author of many articles about .NET development. The author of five technical books, Alessandro was awarded MVP of the Year for Visual Basic in 2009 and 2010.
Dedication

In your whole life, there will be only two people who will never deceive you: your mum and your daddy. This book is dedicated to my mum and daddy, who always helped me, who sacrificed much in their lives to help make me become a man and see me happy. No mere “thanks” will ever be enough to thank you as you would really deserve. Take care.

Who finds a real friend, finds a treasure. My treasure and friend is Nadia. You’re the one who knows when I’m happy, when I’m sad, and what I really need without me speaking a single word. You’re the best friend one could ever have, and I’m lucky to have you by my side. I wish you all the happiness in the world.
I want to thank Neil Rowe, Brook Farling, and all at Sams Publishing for another great experience of working together. Writing a book like this is a very hard work, but working with people like you makes things easier. Thanks!

My deep, special thanks go to Beth Massi from the LightSwitch team at Microsoft. As the technical editor of this book, Beth worked with a great passion and attention through what I wrote and provided great suggestions about what readers needed to know, all the while sharing her real-world experience. From her contribution to this book, I’ve learned more about LightSwitch than I could have ever learned on my own. LightSwitch developers should always remember her words: “In LightSwitch, the only code you write is the only code you could write, the business logic.” You are a great part of this success, Beth.

My thanks go, as well, to the whole Microsoft Visual Studio LightSwitch team for all the amazing and important technical discussions we’ve had. In particular, I want to thank John Stallo, who always answered my technical questions with great care and attention; Joe Binder, for his helpful suggestions about custom shells; Michael Eng, for important discussions about integrating LightSwitch with Team Foundation Server 2010; Karol Zadora-Przyłęcki and Eric Erhardt, whose explanations about threading and shells have been so important. This is a great team, and I have learned a lot from all of you.

A special mention goes to my friend Michael Washington, Silverlight MVP and one of the foremost experts on LightSwitch in the world. He never tires of helping the LightSwitch developer community. We shared many interesting technical discussions, but most important, Michael has in mind the noblest meaning of the word community. Many thanks, Michael.

Many thanks to Mei Liang and Lisa Feigenbaum at Microsoft, who are passionate about their jobs. They have been very helpful to me, especially in connecting with the right people for information.

There are moments in which you get tired of sleepless nights, and good words from a special friend are always important, no matter how many times you see or talk on the phone with them. So, Nicolle Prosser and Karin Meier, thank you!

Thanks to Alfonso Ghiraldini and Francesco Bosticco for their cordiality.

Great thanks also to the Italian subsidiary of Microsoft, which always encourages my community contributions and important works like this. My special thanks to my MVP lead Alessandro Teglia, who always does his job with an uncommon passion.

I would like to thank my everyday friends, who are always ready to encourage me and who share my happiness for my technical successes, even if they are not techies. So, thanks from the bottom of my heart to Roberto Bianchi, Alessandro Ardovini, Francesca Bongiorni, Paolo Leoni, Leonardo Amici, and Sara Gerevini. You are my second family, and I love you all.

Back in August 2010, I and some other fellow MVPs co-founded the one and only Italian community about Visual Studio LightSwitch, called LightSwitch Tips & Tricks (www.lightswitch.it). This has been an opportunity to share discussions on the product and learn something new every day. So, thanks to my fellow Microsoft MVPs Diego Cattaruzza, Renato Marzaro, and Antonio Catucci, but also to my great friend Marco Notari, and to all of you readers who trusted me by purchasing this book and who come to read my blog or my articles daily.
We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we’re doing right, what we could do better, what areas you’d like to see us publish in, and any other words of wisdom you’re willing to pass our way.

As an executive editor for Sams Publishing, I welcome your comments. You can email or write me directly to let me know what you did or didn’t like about this book—as well as what we can do to make our books better.

*Please note that I cannot help you with technical problems related to the topic of this book. We do have a User Services group, however, where I will forward specific technical questions related to the book.*

When you write, please be sure to include this book’s title and author as well as your name, email address, and phone number. I will carefully review your comments and share them with the author and editors who worked on the book.

Email: feedback@samspublishing.com

Mail: Neil Rowe  
Executive Editor  
Sams Publishing  
800 East 96th Street  
Indianapolis, IN 46240 USA

Reader Services

Visit our website and register this book at informit.com/register for convenient access to any updates, downloads, or errata that might be available for this book.
I've had the pleasure of knowing Alessandro for many years, ever since I started my career at Microsoft. The first time I ever heard from Alessandro was through my blog contact form, and we have collaborated on community content and activities ever since. Not long after he published his first article on MSDN, Alessandro was awarded Microsoft Most Valuable Professional (MVP) for his exceptional community leadership and technical expertise. It's always fun working with Alessandro and to see his excitement and passion for the developer community, something we both share.

When Visual Studio LightSwitch was announced and the first beta released back in August 2010, Alessandro was right onboard creating a LightSwitch Tips & Tricks community modeled after his successful Visual Basic Tips & Tricks forums and blogs. His passion for Visual Studio LightSwitch is as strong as his passion for Visual Basic. Alessandro aims to make programming easy, fun, and productive, just as we strive to do on the Visual Studio LightSwitch team. It was an honor for us to have Alessandro as one of our community rock stars and LightSwitch advocates so early on.

At that time, Alessandro started on a journey to write this book and asked me to be the technical reviewer. I was very excited to do it, especially because I had just become the Community Manager for the LightSwitch team and was diving head first into the product myself. I was particularly excited to join the LightSwitch team because I have a long history of building business applications and information systems, particularly for the healthcare industry. I was immediately amazed at what I could build in such a short amount of time with Visual Studio LightSwitch. I was also impressed with all the details it handles automatically for you, such as CRUD plumbing, concurrency handling, dirty checking, automatic screen navigation and lookup lists, user permissions, and so much more.

It doesn’t take long to realize how extremely productive you can be with Visual Studio LightSwitch, regardless of your programming skills. Because of this, and the fact that we made many enhancements from beta 1 to beta 2 to RTM, it was a challenge at times to get the book organized in the optimal way. I think we achieved our goal. The book begins with the beginner in mind and then moves deeper into professional developer topics. I see LightSwitch developers progressing this way, as well, starting off with minimal coding, releasing business productivity applications in no time, and becoming the company hero. Then they may move on to more-advanced customizations as the requirements start to hit the limits of what you can do out of the box. LightSwitch has an extensive extensibility model, and when you need it, you can do just about anything. The end of the book shows you how to take advantage of this.

Alessandro explains Visual Studio LightSwitch in a way that is easy to understand, and his passion comes through in every paragraph. I know you will find this book filled with prescriptive guidance and tips and tricks that you can apply to the LightSwitch business applications you are building today. I really enjoyed reviewing this book, and I’m sure you will enjoy reading it.

—Beth Massi
This page intentionally left blank
Visual Studio LightSwitch is a Rapid Application Development (RAD) environment focused on making it easier to build line-of-business applications (that is, data-centric software). Up to now, you have learned what LightSwitch is and a fair bit about its integrated development environment (IDE), but there is a lot more to learn. Starting from this chapter, you begin developing applications with LightSwitch, and in the process, you learn how easy it is to build complex applications in a very few steps. This is the probably the most important chapter in the book because it provides the fundamentals of LightSwitch development, offering tons of information about creating data sources, creating screens, and implementing data validation, and covering important concepts such as business data types. In this chapter, you create a simple application based on a single data source. This is enough to understand how LightSwitch works and how you can take advantage of all its features to create high-quality, professional business software quickly.

Creating a New Application

This chapter guides you through the process of building an application that enables you to keep track of all your contacts, such as family members, friends, and co-workers. You will be able to store information about each person you add to your contacts list. For example, you might want to add information such as name, phone number, email address, and so on. Although this might look simple, this example teaches you how to use a number of LightSwitch features.
To create a new project, select File, New Project, and in the New Project dialog, select the LightSwitch Application (Visual Basic) project template.

**WHY VISUAL BASIC?**

Code examples in this book are presented in Visual Basic. If you develop in Visual C#, you can download the C# version of the code from this book’s page on the InformIT website. The reason for using Visual Basic is that one of the Visual Studio LightSwitch’s purposes is to make migration to the .NET Framework easier for developers coming from Microsoft Visual Basic 6, Microsoft Access, or Microsoft Visual FoxPro.

Let’s name this new project **ContactsManager**, as shown in Figure 3.1.

---

![Figure 3.1 Creating the new ContactsManager application.](image)

Click OK. After a few seconds, LightSwitch shows the LightSwitch Designer introduced in Chapter 2, “Exploring the IDE.” This is when you decide how to create the data source for the application. As you will learn throughout this book, Visual Studio LightSwitch can create new data sources or grab schemas and data from existing sources such as SQL Server or other databases, SharePoint, or WCF RIA Services. Right now, however, you are at a point when you still have a lot to learn about LightSwitch, so the best approach for a complete understanding is to create a new table from scratch.
Creating a New Data Source

Tables are primitive data containers. A table represents a series of items of the same type. For example, a table can represent a list of customers or of employees or of cars or of any object that has some properties. Tables are divided into rows and columns. A row represents a single item. For example, if you have a table storing a list of customers, a row represents a single customer in the list. In LightSwitch terminology, a single item is referred to as an *entity*, and a list of items is called *entity collection*. Columns represent properties of the elements that the table stores. For example, a customer has a company name, an address, and so on. This information is represented with columns. In LightSwitch terminology, a column is also referred to as a *property*. Actually, columns also allow you to specify the type of the property (such as date and time, strings, numbers), as detailed later in this section. Continuing with the creation of this application, the first step is to create a new entity that will represent a single person in the contacts list. Do so, and name it *Contact*. In the LightSwitch Designer, click *Create New Table*. Doing so opens the Table Designer, as shown in Figure 3.2.

![Table Designer](image)

**FIGURE 3.2** The Table Designer shows a new empty entity.

Here you can design your entity by specifying its properties. By default, the focus is on the entity’s title, which is Table1Item. Rename this *Contact*. 
AVOID CONFUSION BETWEEN TABLES AND ENTITIES

Although the Table Designer looks like a table editor, it actually allows you to design a single entity. This is why the entity name is singular (Contact). After you design your entity, Visual Studio LightSwitch generates a table for you, pluralizing the entity’s name. So, in the current example, the generated table will be named Contacts.

At this point, you can begin designing your new entity by adding properties. Notice that, by default, LightSwitch adds an Id property of type Integer (a type representing integer numbers), which identifies the entity as unique in the table (this is why such a property cannot be edited or removed). Therefore, such a property is an auto-incrementing index that is incremented by one unit each time a new element is added to the table. If you have some experience with other data-access tools such as Microsoft SQL Server or Microsoft Access, you might think of this as an identity field.

Adding Entity Properties

Each property has three requirements: the name, the type, and whether it is required (meaning that the information is mandatory or not). With regard to an entity that represents a contact, the first property you may want to add is the last name. So, click inside the Add Property field under the Name column and type LastName. Remember that property names are alphanumeric combinations of characters and digits and cannot contain blank spaces. If you want to provide some form of separation between words, you can use the underscore (_) character or you can use the so-called camel-case notation, where the words that compose an identifier start with an uppercase letter. LastName is an example of a camel-case identifier. LightSwitch automatically sets labels in the user interface to contain a space between camel-cased words automatically.

After you type the property name, you can press Tab on the keyboard to switch to the Type field. You can see the list of available data types and select a different one by expanding the Type combo box. By default, LightSwitch assigns the String type to each newly added property. Because the last name is actually a string, you can leave unchanged the default selection. Before providing the full list of available data types, let’s focus on the Required field. You mark a property as required when you want to ensure that the user must provide information for that property. In this example, the last name is mandatory because it is the piece of information that allows the identification of a person on our list of contacts, so Required is checked. LightSwitch marks new properties as required by default, so you need to manually unselect the box if specific property information is optional. Figure 3.3 shows the result of the previous addition.

Each entity can expose different kinds of information. This is accomplished by assigning the most appropriate data type to each property. Before adding more properties to the Contact entity, it is important for you to understand what data types are and what data types Visual Studio LightSwitch offers.
Creating a New Data Source

Understanding Data Types
Whatever tasks your application performs, it manipulates data. Data can be of different kinds, such as numbers, strings, dates, or true or false values. When you add a property to an entity, you need to specify its data type to clarify what kind of data that property is going to manipulate. Visual Studio LightSwitch provides some built-in data types that are based on types exposed by the .NET Framework 4.0 and that in most cases are sufficient to satisfy your business needs. Table 3.1 summarizes available data types.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary</td>
<td>Represents an array of bytes</td>
</tr>
<tr>
<td>Boolean</td>
<td>Accepts true or false values</td>
</tr>
<tr>
<td>Date</td>
<td>Represents a date without time information</td>
</tr>
<tr>
<td>DateTime</td>
<td>Represents a date including time information</td>
</tr>
<tr>
<td>Decimal</td>
<td>Represents a decimal number in financial and scientific calculations with large numbers (a range between (-79228162514264337593543950335 ) and (79228162514264337593543950335))</td>
</tr>
</tbody>
</table>
### TABLE 3.1 Available Data Types in Visual Studio LightSwitch

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double</td>
<td>Represents a large floating number (double precision) with a range from $-1.79769313486232e308$ to $1.79769313486232e308$</td>
</tr>
<tr>
<td>Email Address</td>
<td>Represents a well-formed email address</td>
</tr>
<tr>
<td>Image</td>
<td>Represents an image in the form of binary data</td>
</tr>
<tr>
<td>ShortInteger</td>
<td>Represents a numeric value with a range between $-32768$ and $32767$</td>
</tr>
<tr>
<td>Integer</td>
<td>Represents a numeric value with a range between $-2147483648$ and $2147483647$</td>
</tr>
<tr>
<td>LongInteger</td>
<td>Represents a numeric value with a range between $-9223372036854775808$ and $9223372036854775807$</td>
</tr>
<tr>
<td>Money</td>
<td>Represents a monetary value, including the currency symbol and the appropriate punctuation according to the local system culture</td>
</tr>
<tr>
<td>Phone Number</td>
<td>Represents a well-formed phone number according to U.S.-supported formats</td>
</tr>
<tr>
<td>String</td>
<td>Represents a string</td>
</tr>
<tr>
<td>Guid</td>
<td>Represents a <em>globally unique identifier</em>, which is a complex, randomly generated number typically used when you need a unique identifier across different computers and networks</td>
</tr>
</tbody>
</table>

Using the appropriate data type is very important because it makes data manipulation easier, provides the right mapping against the back-end database, and in most cases can save system resources. As an example, instead of using a `String` to represent a date, you use the `Date` type. This is useful (other than natural) because SQL Server has a corresponding data type and the .NET Framework provides specific objects for working with dates if you need to write custom code. The same consideration applies to numbers, Boolean values, and binary data.
**MAPPING OTHER .NET DATA TYPES**

Table 3.1 summarizes data types that Visual Studio LightSwitch supports when adding new entities. By the way, to provide the best experience possible, when importing existing data from external data sources (such as SQL Server databases), LightSwitch maps imported types into a more convenient .NET type, even if this is not generally available in the Entity Designer. For a better understanding, consider the Northwind database, which is a free sample database from Microsoft popular in the developer community (see http://archive.msdn.microsoft.com/northwind). If you consider such a database, the Picture column in the Category table is of the SQL type IMAGE, but LightSwitch does not map it as an Image. Instead, it maps it to a Binary type (which is an array of System.Byte in .NET), but you are still allowed to change it into an Image. Another example is the Discount column in the Order_Details table; this is of the SQL type REAL, but it is mapped to a Single .NET type. In this case, the mapping cannot be changed because this is a primitive type. Generally, you can leave unchanged the default mapping because LightSwitch takes care of translating data for you into the most appropriate form.

**PREFER INTEGER FOR NUMBERS**

With regard to numbers, both the Visual Basic and Visual C# compilers are optimized to work with the Integer data type. For this reason, you should always prefer Integer even when an integer number is in the range of ShortInteger. For the same reason, you should use LongInteger only when you are sure that your application will work with numbers greater than the range supported by Integer.

**The Concept of Business Data Types**

In Table 3.1, you might have noticed something new in the LightSwitch approach to data types. In contrast to other development tools and technologies, including .NET Framework 4 and Visual Studio 2010, LightSwitch introduces the concept of business data types. For example, suppose you want to add a property to an entity for inserting or displaying monetary information. Before LightSwitch, this was accomplished by using the Decimal data type, which can display decimal numbers and is therefore the most appropriate .NET type in this scenario. This is correct but has some limitations: For example, you must write some code to add the currency symbol or format the information according to the local system culture. Next, consider email addresses. Developers usually represent email addresses with the String data type but they have to implement their own validation logic to ensure that the string is a well-formed email address. The same is true for phone numbers. This approach makes sense in a general-purpose development environment such as Visual Studio 2010. But Visual Studio LightSwitch is a specialized environment
focusing on business applications, so the LightSwitch team at Microsoft introduced four new data types, specifically to solve business problems:

- **Money**, a data type for displaying currency information that provides the appropriate currency symbol and culture information according to the user's system regional settings.
- **Image**, which allows storing and returning an image in the form of binary data.
- **Email Address**, which accepts only valid email addresses and implements validation logic that throws errors if the supplied email address is not well formed.
- **PhoneNumber**, which accepts only valid phone numbers and which implements validation logic that throws errors in case the supplied phone number is not well formed. This data type can be customized to accept phone numbers in a format different from the default one, built specifically for the United States.

If you think that any data-centric applications must validate the user input to ensure that the entered data is valid, business data types can save you a lot of time, especially because you do not need to write the validation logic: LightSwitch does it for you. You will get a visual sensation of the power of business data types in this chapter when you run the application and test the validation functionalities. In addition, Visual Studio LightSwitch provides an extensibility point where you can add custom business types, as described in Chapter 19, “LightSwitch Extensibility: Data and Extension Deployment.” Now you know everything important about data types in LightSwitch and are ready to complete the design of the Contact entity by adding specialized properties.

### Building a Complete Entity

So far, the Contact entity exposes only the LastName property, so it needs to be extended with additional properties that complete the single contact information. Table 3.2 shows the list of new properties (and their data type) that are added to the entity. You add properties by clicking inside the Add Property field and specifying types from the drop-down Type combo box, as you learned earlier.

<table>
<thead>
<tr>
<th><strong>Property Name</strong></th>
<th><strong>Type</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>FirstName</td>
<td>String</td>
</tr>
<tr>
<td>Age</td>
<td>Integer</td>
</tr>
<tr>
<td>Address</td>
<td>String</td>
</tr>
<tr>
<td>City</td>
<td>String</td>
</tr>
<tr>
<td>Country</td>
<td>String</td>
</tr>
<tr>
<td>PostalCode</td>
<td>String</td>
</tr>
</tbody>
</table>
Creating a New Data Source

TABLE 3.2 Properties That Complete the Contact Entity

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email</td>
<td>Email Address</td>
</tr>
<tr>
<td>HomePhone</td>
<td>Phone Number</td>
</tr>
<tr>
<td>OfficePhone</td>
<td>Phone Number</td>
</tr>
<tr>
<td>MobilePhone</td>
<td>Phone Number</td>
</tr>
<tr>
<td>Picture</td>
<td>Image</td>
</tr>
<tr>
<td>JobRole</td>
<td>String</td>
</tr>
</tbody>
</table>

None of the new properties is required because the entity already provides two ways to identify a contact as unique: the Id property and the LastName property. After you complete adding properties, the Contact entity looks like Figure 3.4.

FIGURE 3.4 The Contact entity has been completed.

The Contact entity represents a single contact, but Visual Studio LightSwitch also generates a Contacts table that represents a list of items of type Contact. If you open Solution Explorer and expand the Data Sources node, you can see how such a table is visible, as
demonstrated in Figure 3.5. If you name your entities with words from the English language, LightSwitch automatically pluralizes the entity’s name when generating the table, thus providing the appropriate name for the new table.

![Solution Explorer](image)

FIGURE 3.5 The new Contacts table is visible in Solution Explorer.

Now that you have successfully created a new entity and a table for storing a list of entities, you might wonder where data is stored by your application. This is briefly explained in the next subsection and is revisited in Chapter 12, “Dissecting a LightSwitch Application.”

**Data Storage**

LightSwitch applications use the Microsoft SQL Server database engine to store their data. In fact, one of the prerequisites when you install LightSwitch is the presence on your development machine of SQL Server Express Edition. Supported versions are 2005, 2008, and 2008 R2. When you create an application, Visual Studio LightSwitch creates a SQL Server database for the application itself, also known as the *intrinsic database*, naming such a database ApplicationDatabase.mdf. This is the physical storage for your data until you run the application on the development machine. When you deploy the application to a web server or a networked computer, you also need to deploy the database. You can decide to export the current database and attach it to the running instance of Microsoft SQL Server, and its name is replaced with the name of the application (for example, ContactsManager.mdf). Alternatively, you can let the deployment process build and attach to SQL Server a brand-new database with a different name, but you can also publish the database in the cloud by deploying it to your SQL Azure storage. LightSwitch can perform these tasks for you, especially if you use the one-click deployment systems covered in Chapter 10, “Deploying LightSwitch Applications.” For now, you just need to know that data is stored into ApplicationDatabase.mdf, which is located in the %ProjectFolder%\Bin\Data folder.

**The User Interface: Implementing Screens**

The Contact entity and the Contacts table represent your data structure. A professional standalone application needs a graphical user interface that enables users to enter and retrieve data easily from the database. In LightSwitch applications, the user interface has the form of *screens*, which are what users see. You may think of screens as of windows
made of fields you fill in with data, of grids showing lists of data, and of buttons (or other controls) bound to perform some actions such as saving or searching data. You do not need to build screens manually because LightSwitch ships with a number of full-feature, reusable screen templates that satisfy most of the needs in a business application. Such templates implement fields, buttons, grids, and all other user interface elements required to work with data. Default command buttons are organized inside tabs exposed by a Ribbon Bar control (which mimics the Microsoft Office user interface). If you ever developed applications with Microsoft Access, you can easily compare LightSwitch screens to Access forms. Offering predefined screen templates is another benefit in Visual Studio LightSwitch because these enable you to quickly build the user interface for an application without writing a single line of code or without any work from the developer. At a higher level, you can customize screens as follows:

- Rearrange predefined controls, moving them to different positions on the screen
- Add tabs and buttons
- Style screens with different themes
- Add custom Silverlight controls
- Delete the entire content tree and add data items manually or create an empty screen and still add items manually (Chapter 8, “Aggregating Data from Different Data Sources,” provides an example)

So, what you basically cannot do is lay out the screen pixel by pixel, as you would do in other development environments, but this makes sense in LightSwitch development: The goal of LightSwitch is to provide tools to build fully functional business applications in the quickest way possible, extending such an experience to novices and nonprofessional developers, as well. The idea is that you do not want to (or need to) waste your time in building the user interface; you just take advantage of what LightSwitch has ready to run. Visual Studio LightSwitch currently offers five screen templates. In this chapter, you use only two of them. In Chapter 4, “Building More-Complex Applications with Relationships and Details Screens,” you begin to work with the other three templates. The reason for this is that in this chapter, you are working with a single table database and therefore you will learn about screens that work well in this scenario. This approach is also useful for understanding how screens work, how you can customize their properties, and how binding data to screens is really straightforward. In Chapter 4, you learn how to build master-details applications, and thus you will also learn about screens that enable you to work with one-to-many relationships. In the current scenario, our users need a way to add new contacts to the database and one to show the list of available contacts.

**Controls Overview**

The user interface of any application is made of controls.

Each control is a graphical element that the user can easily associate with a particular action in the application. Buttons and text boxes are examples of controls. These are typically arranged within containers (or panels). For example, the main application’s window
is the root container for nested control containers and controls. In Visual Studio LightSwitch, screens are made of controls, too. Because screens are based on a predetermined set of templates, LightSwitch offers a common set of controls that you interact with at both design time and at runtime. This section provides a brief overview of common controls in LightSwitch so that you can have a general idea about which elements are actually used within screen templates. Table 3.3 summarizes controls in LightSwitch.

### TABLE 3.3 Common Controls in LightSwitch

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
<th>Supported Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Screen Command Bar</td>
<td>A control that replicates the Microsoft Office Ribbon UI and that is a container for buttons organized in tabs</td>
<td>At the screen level</td>
</tr>
<tr>
<td>Button</td>
<td>A clickable button that executes a particular action</td>
<td>At the screen level</td>
</tr>
<tr>
<td>TextBox</td>
<td>A field where the user can enter text</td>
<td>String, ShortInteger, Integer, LongInteger, Double, Decimal, Guid</td>
</tr>
<tr>
<td>Date Picker</td>
<td>A control for easily selecting dates</td>
<td>Date</td>
</tr>
<tr>
<td>Date Viewer</td>
<td>A control that displays date according to specific formatting rules</td>
<td>Date</td>
</tr>
<tr>
<td>DateTimePicker</td>
<td>Works like the Date Picker but also allows users to select a time of day</td>
<td>DateTime</td>
</tr>
<tr>
<td>DateTimeViewer</td>
<td>Works like the Date Viewer but also displays a time of day</td>
<td>DateTime</td>
</tr>
<tr>
<td>Label</td>
<td>A control used for displaying simple text messages</td>
<td>String, ShortInteger, Integer, LongInteger, Double, Decimal, Guid</td>
</tr>
<tr>
<td>Image Editor</td>
<td>A control used for uploading images to the application</td>
<td>Image</td>
</tr>
<tr>
<td>Image Viewer</td>
<td>A control able of displaying images stored inside the database</td>
<td>Image</td>
</tr>
<tr>
<td>Email Address Editor</td>
<td>A special text box used to enter and validate email addresses</td>
<td>Email Address</td>
</tr>
<tr>
<td>Email Address Viewer</td>
<td>A special control used to display well-formed email addresses</td>
<td>Email Address</td>
</tr>
<tr>
<td>PhoneNumber Editor</td>
<td>A specialized text box to enter, validate, and format phone numbers</td>
<td>Phone Number</td>
</tr>
</tbody>
</table>
TABLE 3.3 Common Controls in LightSwitch

<table>
<thead>
<tr>
<th>Control</th>
<th>Description</th>
<th>Supported Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phone Number Editor</td>
<td>A read-only text box that displays phone numbers according to specific formatting rules</td>
<td>Phone Number</td>
</tr>
<tr>
<td>DataGrid</td>
<td>Shows the content of data coming from an entity collection in a tabular representation</td>
<td>Entity collection</td>
</tr>
<tr>
<td>DataGridRow</td>
<td>Represents a single row (that is a single entity) in a DataGrid</td>
<td>Single entity</td>
</tr>
<tr>
<td>List</td>
<td>Shows the content of data coming from an entity collection under the form of a scrollable list</td>
<td>Entity collection</td>
</tr>
<tr>
<td>Money Editor</td>
<td>A specialized text box where you can enter monetary information</td>
<td>Money</td>
</tr>
<tr>
<td>Money Viewer</td>
<td>A read-only text box specific for displaying monetary information, including the currency symbol and localized information such as the symbol used for the decimal point</td>
<td>Money</td>
</tr>
<tr>
<td>CheckBox</td>
<td>A control that allows users to select a true or false condition</td>
<td>Boolean</td>
</tr>
</tbody>
</table>

This set of controls is the most common in LightSwitch development and covers the needs for almost any business application. Of course, you can create and add custom Silverlight user controls, which you learn in Chapter 7, “Customizing Applications with Buttons, COM Automation, and Extensions.” You first need to understand how to create, manage, and customize screens. Let’s begin by creating a data entry screen, which is the place where users enter data and is also the first contact you have with screens in LightSwitch.

Creating a Data Entry Screen

To add a screen to the project, you use one of the following options:

- Right-click the Screens folder in Solution Explorer and select Add Screen.
- Click the Screen button in the Entity Designer.
- Select Project, Add Screen.

Whichever option you use, you are prompted to specify a screen template, a screen name, and screen data in the Add New Screen dialog, shown in Figure 3.6.
FIGURE 3.6 You can add new screens to the project in the Add New Screen dialog.

On the left side of the dialog, you see the list of available templates. Select the New Data Screen one. Notice that in the center of the dialog, you can see a preview of what the screen looks like. Next, you need to specify the data source that will be associated with the screen. In the Screen Data combo box, pick up the table you want to associate with the screen. Basically, here you see the list of all the entities you defined in your project. Currently, there is only the Contact entity, so just select this one. After you select the screen data, a check box becomes visible. In this case, it is named Contact Details and, if you check it, the resulting application will also allow editing the entity’s details. When you choose the data source, LightSwitch automatically changes the content of the Screen Name text box, providing a more meaningful value (CreateNewContact) in the current example. Now click OK. At this point, Visual Studio LightSwitch switches to the Screen Designer view. As explained in Chapter 2, you will not get an interactive designer for the screen; instead, you get a list of controls laid out in a content tree comprising the screen’s user interface. Figure 3.7 shows the Designer screen for the CreateNewContact screen.

WHERE’S THE XAML? A NOTE FOR WPF AND SILVERLIGHT DEVELOPERS

If you have had any experience developing Windows Presentation Foundation (WPF) and Silverlight applications, this is the point at which you might wonder where the user interface elements are actually declared. In WPF and Silverlight, you define the user interface via the Extensible Application Markup Language (XAML), and you would probably expect that this happens in LightSwitch, too. In LightSwitch applications, you have no XAML. The user interface (and the entire visual tree) is generated and compiled for you when you build the application.
The User Interface: Implementing Screens

This also ensures that nobody can accidentally make changes to the user interface that might prevent the application from running correctly. This might seem disappointing if you are an expert developer, but it makes sense because LightSwitch is intended for both experts and novices.

FIGURE 3.7 The Screen Designer for the CreateNewContact screen.

The user interface composition is shown in a hierarchical fashion. At the top is an element named Rows Layout that specifies how the user interface is globally organized. Nested are the Screen Command Bar (that is, the Ribbon) with two buttons (one for saving data and one for refreshing controls) and an element named Rows Layout that contains all controls that allow user input. In particular, you can see a number of text boxes that accept input from the user, a control named Email Address Editor that is specific for accepting email addresses, a Phone Number Editor control that can accept and format phone numbers, and an Image Editor control that allows uploading an image. Rows Layout consists of two containers of controls that are part of a major set that is described later. For now, you need to understand how the user interface is organized and that you have no form designer as in Visual Studio 2010. It is worth mentioning that the new screen now appears in Solution Explorer (see Figure 3.7). On the left side of the designer, you have an element named ContactProperty that represents a single contact on the screen. This element is expanded to show the data properties, which basically are the same properties you defined in the Contact entity. There are also some elements referring to buttons (Close, Refresh, and Save).
There is nothing else to do. With a simple sequence of mouse clicks, you successfully added a new screen to your project. But adding data is not the only requirement in this application; we also need to enable users to display and search available data.

Creating a Search Screen

Every business application must provide some user interface for displaying the data stored inside the database. In addition, it should enable users to easily search data. Visual Studio LightSwitch provides a screen template that includes such requirements. Open again the Add New Screen dialog and select the Search Data Screen template (see Figure 3.8).

Select the Contacts data from the Screen Data combo box so that the Screen Name box’s content is also updated to SearchContacts. Notice that you have now specified a table rather than a single entity. This is because the search screen is for showing a list of entities.
TABLE SELECTION

Whatever screen template you add to your application, selecting the table data is always accomplished via the Screen Data combo box.

After you click OK, the Screen Designer displays the composition of the search screen, as shown in Figure 3.9.

As you can see, the screen consists of the following components:

- A container of type Rows Layout, which allows arranging nested controls.
- The Screen Command Bar element, representing the Ribbon user interface, with nested buttons.
- A Data Grid control, which provides the ability of displaying tabular data.
- Nested controls such as a Command Bar, which contains action buttons, and a Data Grid Row element. This represents a single entity (one contact in our scenario), and
it is made of several nested elements, each representing a field in the contact definition. In particular, you can see labels and specific viewers (Email Address Viewer, Image Viewer, and Phone Number Viewer) for business data types.

On the left side of the designer, you can see the data source composition. There is a Contacts element that, as its name implies, represents a collection of items of type Contact. Basically, a collection of items is nothing but a .NET representation of a database table. The Contacts object is the data source for the screen, and each item is data-bound to a single row on the screen itself. LightSwitch performs this work for you. Then the Selected Item property from the collection represents a single entity that is selected within the search screen at runtime, and this is why in the designer, you see the list of properties exposed by your entity. Now that you have added the second necessary screen, the application is ready to run. In a few minutes, you have built a business application that can add, edit, and display your favorite contacts. Now it is time to see how your application works.

Testing the Application on the Development Machine

LightSwitch applications can be executed on the desktop or inside a web browser. This is a tremendous benefit because you can make your programs available as desktop or web applications just by changing one setting. In this chapter, you see how to run your program both as a desktop client and as a web application, starting from the default setting that is the desktop client.

SILVERLIGHT 4 UNDER THE HOOD: OUT-OF-BROWSER APPLICATIONS

You learned in Chapter 1 that LightSwitch applications are nothing but applications built on Silverlight 4. Starting from version 3, Silverlight introduced a feature known as out-of-browser applications, which allows installing and running a Silverlight application as if it were a classic desktop client program. Out-of-browser applications run effectively as desktop clients, so they have more permission over the target machine than web clients. When you create a new LightSwitch application, it runs by default as a 2-tier desktop client. This is actually an out-of-browser Silverlight application unless you change a specific setting in the Application Designer, as demonstrated later in this section. Another great benefit is that (and in contrast to the classic Silverlight development, where you need to manually set some options) in LightSwitch, you just need one mouse click to switch between desktop clients and web clients, and this can be done as many times as you want during the application development process. It is important to emphasize that most of the power of your LightSwitch applications results from the Silverlight infrastructure that runs behind the scenes.
Starting the Application as a Desktop Client

RUNNING FROM WITHIN LIGHTSWITCH

When you run applications from within the LightSwitch development environment, the IDE automatically attaches an instance of the debugger to the application. In other words, applications running from within the IDE are always in debugging mode (in contrast to other editions of Visual Studio 2010, in which you can switch from the debug configuration to the release configuration). Actually, this is not a problem. You will read more about debugging in LightSwitch in Chapter 14, “Debugging LightSwitch Applications”; this current chapter focuses instead on how the application works based on entities and screens you build.

To run the application, press F5. After a few seconds, it displays what you see in Figure 3.10.

![The application is finally running.](image)

Before going into detail about how the application works, there are a few things to point out:

- The application implements a Ribbon Bar showing a built-in tab (Home) that includes two default buttons: Save and Refresh. This piece of the user interface is
common to all screens and can be customized by adding new buttons or new tabs via the Screen Designer or in custom code.

- There is a Menu bar that is common to all screens and that includes collapsible panels nesting common shortcuts. By default, LightSwitch creates a Tasks panel that lists available screens and that you use for navigation between screens.

- Names of items in the user interface, including data fields and shortcuts, are auto-generated and are taken from tables and screen definitions.

- The remaining client area of the application is about screens. In Figure 3.10, you see the Create New Contact screen, which you use to add a new contact. Screen navigation is organized in tabs. If you click Search Contacts in the Tasks panel, another tab shows the Search Contacts screen (see Figure 3.11).

![Figure 3.11 Screens are organized in tabs.](image)

The user interface is intelligent enough to understand that the search screen is not used for data editing, so here the Save button is not available. It is also worth mentioning that what you are now seeing on screen has been created without writing one line of code, because LightSwitch takes care of all the plumbing for you. Let’s now see how the application works by adding/editing data and displaying lists of data.
REFERRING TO SCREEN NAMES

In this book, we refer to screen names by invoking either the name as it appears in the
designer or the name that is generated in the user interface. This means that
SearchContacts and Search Contacts (the latter has a blank space in the middle) are
two ways to refer to the same screen. Typically, the first way is used when referring to
the screen while designing the user interface, whereas the second way is used when
running the demo applications.

Adding and Editing Data

To add new data, you can either click Create New Contact in the Tasks panel or on the
Create New Contact tab. This screen opens first because it was the first to be created. If
you want to change the startup screen, you can go to the Application Properties and select
the Screen Navigation tab. This feature is discussed in detail in Chapter 4.

You just fill in fields according to the information you want to add. Notice that required
fields are presented in bold (such as Last Name in the current example). To fill the Picture
field (and ones like it), simply pass the mouse over such a field and click the green arrow
to add the picture from disk or the click the black cross to remove an existing picture from
the field. Figure 3.12 shows what the screen looks like when filled with custom data.

![Figure 3.12: Filling the screen with information for a new contact.](image-url)
After you have finished adding information, click **Save** to permanently store the information to the database. When you save your data, the application shows an edit screen for the newly added item.

The star symbol also disappears, and the tab title is replaced with the content of the Last Name field. (This automatic choice is because it was the first property added to the entity.) Finally, the Id read-only field shows the record number of the element inside the database. Figure 3.13 shows what the screen looks like after saving data.

![Figure 3.13 The screen changes after saving data.](image)

Adding and Formatting Phone Numbers

While entering data, you might have noticed that fields of type *Phone Number* were automatically formatted. The *Phone Number Editor* control in LightSwitch applications can parse phone numbers and present them in one of the built-in formats if they are recognized as valid; otherwise, data remains as you entered it. The default formatting can be overridden by passing the mouse over the field and clicking the down arrow on the right
side of the field. This shows a group of fields that you fill with your local phone number information, as shown in Figure 3.14.

![Figure 3.14 Providing local information for phone numbers.](image)

In particular, you can specify the following information:

- **Country Code**: The phone number prefix for your country
- **Area Code**: The local prefix for your area
- **Local Number**: The actual phone number without any prefixes
- **Extension**: Additional information for the phone number

After you specify such information, the Phone Number field is updated to reflect changes. Sometimes the built-in phone number formats are not enough to satisfy your requirements, especially if you do not live in the United States. Fortunately, LightSwitch enables you to add custom phone number formats and edit built-in formats, as explained in the next subsection.

**Customizing Phone Number Formats**

The Phone Number Editor control provides automatic phone number formatting capabilities by comparing the phone number entered by the user with one of the built-in phone
number formats available for the Phone Number data type. If the phone number entered by the user matches one of the available formats, the Phone Number Editor presents the number according to this format. You can customize phone number formats by both editing built-in formats and providing new ones. To accomplish this, you need to close the application and open the Table Designer for the desired entity (Contact in the current example). Once the entity is shown in the designer, select one of the properties of type Phone Number. Continuing the current example, select the HomePhone property, and then press F4 to enable the Properties window. Figure 3.15 shows what this window looks like with regard to the selected property.

FIGURE 3.15 The Properties window for properties of type Phone Number.

FIRST CONTACT WITH THE PROPERTIES WINDOW

This is the first time you use the Properties window in this book. Basically, this is the place where you can assign all customizable properties for a data type with the appropriate values. In most cases, customization possibilities are self-explanatory (such as Maximum Length). In other cases, however, explanations are required. You will use the Properties window many times throughout this book, and you will see it in action with almost all available data types and in the most common scenarios (with all the necessary explanations where needed).
Among the available properties, you can find a shortcut named Phone Number Formats. If you click it, the Phone Number Formats dialog appears, showing all the available built-in phone number formats, as shown in Figure 3.16.

![Phone Number Formats dialog](image)

**FIGURE 3.16** The Phone Number Formats dialog enables you to customize phone number formats.

To edit an existing format, just click the box related to the format and replace it with custom information, remembering that:

- The letter **C** represents the country code. You can enter up to three **C** letters for country codes requiring three numbers.
- The letter **A** represents area code numbers. There is basically no limit to the number of **A** letters.
- You can enclose letters between parentheses to provide this kind of formatting.
- The letter **N** represents the local number. You must specify the exact number of **N** letters to match a fixed phone number length. For instance, you add six **N** letters if you want to match a phone number exactly six characters long, but not numbers that are seven or five characters long.

You can test whether a phone number matches one of the listed formats by typing it inside the Test a Phone Number text box. This is useful to understand how formats work. You follow exactly the same rules if you want to specify a new custom format. Just click inside the Add Format field at the bottom of the list and begin writing your format by following the explanations provided in the previous bulleted list. Notice that Visual
Studio LightSwitch validates your format while you write it. If it does not comply with built-in rules, LightSwitch shows validation error messages, as shown in Figure 3.17 with regard to C letters.

![Image of LightSwitch validation](image)

**FIGURE 3.17** LightSwitch performs validation when you are adding a custom format.

After you finish adding your custom phone number format, click OK. If the user enters a phone number that matches your format, it is formatted according to your rules.

**Automatic Check for Pending Changes**
The LightSwitch infrastructure automatically provides to applications the so-called *dirty checking* feature, which is the ability to check whether the user is attempting to close a screen or the application without saving pending changes (and ask for confirmation that the user really wants to do this). For example, if you are adding or editing a contact and then try to close the CreateNewContact screen before saving your changes, the application asks whether you want to save or discard changes before closing, as shown in Figure 3.18.

Similarly, users are asked for confirmation when attempting to close the application but there are some unsaved changes. It is important to underline that such a feature is implemented automatically, meaning that you do not need to enable it manually or write a single line of code.

**Displaying and Searching Data**
So far, you have seen how to add items to and edit items in the application’s database via the user interface of the ContactsManager application example. Now it is time to see how the application can list and search existing data via the search screen implemented earlier. Let’s suppose you have added a number of contacts to the application. Now, in the Tasks panel, click the **SearchContacts** element. You will get the full list of data in a tabular
form, similar to what you would get in Microsoft Excel. Figure 3.19 shows an example taken from my machine, showing a list of friends of mine.

**FIGURE 3.18** The application asks for user confirmation before discarding unsaved changes.

**FIGURE 3.19** The search screen displays the full list of data.
As you can see, the tabular representation shows all the available information organized in columns, one per property exposed by the `Contact` entity. By default, the search screen is read-only, so you cannot directly edit items inside the grid (although there are two ways to accomplish editing that are discussed later). Data is shown via a `DataGrid` control; and phone numbers, email addresses, and images are displayed respectively via `PhoneNumberViewer`, `EmailAddressViewer`, and `ImageViewer` controls. The search screen also offers lots of interesting built-in functionalities that the LightSwitch infrastructure offers without the need to write a single line of code, as described in the following sections.

**Paging**

Suppose that you have added thousands of records to your application’s database. If a search screen had to show them all, it would have to load all items in memory and then render the list to the user interface, and this would heavily affect performances. In addition, scrolling the list of records on the screen would also be slow because the user interface would refresh each time you scrolled the list. Because of this, LightSwitch applications offer a built-in paging mechanism. Thanks to paging, the application divides data into virtual pages and loads and displays 45 items at a time. For example, when you run an application that stores 100 elements, it divides the data into 3 virtual pages (45 + 45 + 10) and then it loads and displays the first 45 items. Then you can go to page 2 or 3 (and then go back) by using the `DataPager` control shown in Figure 3.20.

![Figure 3.20](image)

**FIGURE 3.20** The `DataPager` control allows moving between data pages.

The first button on the left side of the control brings you back to the first page, and the last button on the right side of the control moves to the last page. Intermediate controls browse data by one page. Only when you click one of these buttons does the LightSwitch application load and display the corresponding number of items. This paging mechanism is a convenient way to maintain optimal performances and is something that is provided by default, without writing a single line of code. The page size is not fixed; you can replace the default value of 45 with a custom one. To do so, open the Screen Designer and select the collection in the upper-left corner (which is Contacts in the current example). Then, in the Properties window, edit the value of the No. of Items to Display per Page property.

**Searching and Filtering**

As its name implies, a search screen not only displays the list of items from a table, but it also provides search tools. By default, search screens in LightSwitch applications implement a search text box where you can type your search key. Then you click the button with a lens inside or you simply press Enter. For example, you might want to retrieve a specific record from the list. Figure 3.21 shows how to retrieve a contact by its last name.
FIGURE 3.21 Searching for a specific record.

You can simply return to the full list by clicking the button with a blue X near the search box. It is important to mention that the search tool filters all the items that contain the specified text in every string column. For example, if you use the “cat” search key in the sample application, the search screen returns all the items that contain the specified key, and in this case, this is true in the Last Name column, as shown in Figure 3.22.

FIGURE 3.22 Filtering items: first example.
Similarly, if you type the “de” key search, the search screen filters all the items that contain the search key in every string column that in the current example is contained in the Job Role column, as shown in Figure 3.23.

![Figure 3.23 Filtering items: second example.](image)

**Refreshing the Search Results**

Search screens provide a Refresh button on the Ribbon Bar.

You use this button to reload the full list of data, and it is useful when you add a new or edit an existing item. In fact, when you are editing items, the search results do not automatically reflect changes, so you need to refresh the list with the Refresh button. To understand how this works, follow these steps:

1. In the Tasks panel, click **Create New Contact**.
2. Add a new contact by specifying real or sample information, and then save your changes.
3. Return to the search screen.

At this point, you might notice that the new contact is not automatically displayed in the search results. The reason is that the screen displays a snapshot of items stored in the database at the time you executed the search, which is similar to requesting a web page in your browser.

So, click the **Refresh** button on the Ribbon Bar to get the list updated with the new data.

**Editing Data from the Search Screen**

By default, you cannot edit data directly inside cells. By the way, LightSwitch presents cells in the first column (for each row) with a hyperlink that you can click to edit the
Testing the Application on the Development Machine

current item. This refers to the summary property of your table, which by default is the first property you added to your entity and that is used as the default column. You can change this in the Table Designer.

Continuing the previous example, the Contact column refers to the LastName property in the Contact entity, and you can click the last name of each contact to easily edit the item. Figure 3.24 shows what happens when you click the last name of a contact.

As you can see, a new tab appears containing a screen that allows editing the item you selected. Here you just make your changes and then click Save when you are done. It is worth mentioning that LightSwitch created this editing screen for you, saving you a lot of time. You can return to the search results by closing the current tab or by clicking the Search Screen tab.

**Understanding Summary Properties and the Show as Link Feature** Earlier you saw how LightSwitch presents the first property added to the entity with a hyperlink in search screens. By clicking this hyperlink, you can immediately access detail properties for the selected item (see Figures 3.19, 3.22, and 3.23). In the current example, the LastName property value is presented with a hyperlink because that was the first property added to the entity. Of course, you can change the default setting, and this is an easy task. For instance, with regard of the current ContactsManager application, you might want to
make the Email property the hyperlink. If you just want to change the property that is displayed as a hyperlink, in the Screen Designer you simply select the control mapped to the property, and in the Properties window you check the Show as Link check box, as shown in Figure 3.25.

Notice that selecting a new property does not automatically deselect the previous one (in this case, LastName). Therefore, LightSwitch allows you to present multiple properties as hyperlinks, but you have to manually uncheck the Show as Link check box for undesired properties. If you now run the application, you can open contact details by clicking the Email hyperlink. Using Show as Link is the simplest way possible to change the default behavior, but in some situations, this is not enough. In fact, some screens use the Summary control to display a list of items but present them via only one property. For a better understanding, consider Figure 3.26, which is taken from the sample application that will be built in Chapter 4. Notice how the Customer property shows a list of customers but only the company name is displayed. That is an example of Summary control, and the property that is displayed is referred to as a summary property.

By default, the summary property is the first property added to the entity. To change the summary property, double-click the desired table in Solution Explorer, and then in the Table Designer, click the table name. In the Properties window, you can see at this point the Summary Property field, where you can select a different entity property to identify a list of items within a Summary control. With regard to customers, you could select the email address or the name of the contact. Figure 3.27 demonstrates how to change the current summary property.
FIGURE 3.26 Summary properties enable you to display one entity property within the Summary control.

FIGURE 3.27 Setting summary properties.
TESTING SUMMARY PROPERTIES

Generally, search screens are not the place in which summary properties are used. By the way, you could replace the Data Grid control in a search screen with a List control. This uses a Summary control that points to summary properties. You can test this yourself with the current sample application.

Using Controls That Support Editing

The default behavior for search screens is that they are read-only and that you cannot directly edit data within cells, but you can access the edit screen by clicking the hyperlink provided for each row. If you do not like to invoke an external screen for editing data, you can override the default behavior by changing the Use Read-Only Controls property for the screen.

EDITABLE GRID SCREEN

LightSwitch also offers a screen template called Editable Grid, which works exactly like a search screen but that also allows editing data within cells. It is discussed further in Chapter 4. For now, though, focus your attention on modifying properties of a search screen.

To accomplish this, follow these steps:

1. Close the application if it is running.
2. In Solution Explorer, double-click the SearchContacts screen and then press F4 to enable the Properties window.
3. Uncheck the Use Read-Only Controls check box, as shown in Figure 3.28.

FIGURE 3.28 An unchecked Use Read-Only Controls check box.
If you now rerun the application and open the search screen, you can click inside the cells you want to edit and type your information without opening an external screen. Figure 3.29 shows an example, where the Address property for the first contact in the list is being edited.

![FIGURE 3.29  Editing data directly inside the cells.](image)

You can also edit email addresses, phone numbers, and images. This is because the viewer controls (for example, Image Viewer) are replaced with the editor counterparts (for example, Image Editor). Of course, you can still edit data by clicking the hyperlink in each row, which points to the edit screen for the current item. Editing data inside cells is a plus.

## Exporting Data to Microsoft Excel

In most business applications, interaction with the Microsoft Office suite is a common scenario. For example, you might want to export a report to a Microsoft Excel spreadsheet, or you might want to produce letters in Microsoft Word starting from a list of customers. One goal of Visual Studio LightSwitch is to make Office integration easier. Because exporting data to Microsoft Excel is a common scenario, the LightSwitch infrastructure provides automation for this particular task from within search screens. Under the search box is an Export to Excel button, identified with the typical Excel icon. If you click this button, after a few seconds your search results are exported into a new Excel workbook, as shown in Figure 3.30.
The generated Excel workbook shows a list of columns representing exactly the columns in the Data Grid, including column headers. Below, there is the actual list of data that you can treat as you would in any other Excel spreadsheet. For example, you might want to format column headers in bold, but you can also add formulas, edit cells, and so on. Just notice that with regard to pictures, there is no Excel counterpart for the Image data type in LightSwitch, so the Picture column just reports the string representation of the .NET data type that handles images, which is `[Image]` (`System.Byte` is the .NET representation of the type.) Excel integration is, without a doubt, a requirement in most cases, and LightSwitch makes this easy.

**MICROSOFT OFFICE INTEGRATION**

You might wonder what other built-in possibilities LightSwitch offers for integrating applications with Microsoft Office, other than exporting to Excel. The answer is: none, out of the box. You can still automate Office applications by taking advantage of different techniques (such as COM interoperability, which is covered in Chapter 7, Chapter 8, and the Office Integration Pack, which is covered in Chapter 17), but all these techniques require writing code and that you understand some important .NET programming concepts.

**Basic Screen Customizations**

So far, you have generated a fully functional LightSwitch application without making any changes to the user interface. You can customize screens in a number of ways, both basic and advanced. In this chapter, you learn about basic customizations. In Chapter 4, you learn about advanced customizations.
Customizing Display Names and Descriptions
When generating screens, LightSwitch automatically provides descriptive text for controls by picking up the name of the related entity’s property. For example, you have a LastName property in the entity. When you create screens, LightSwitch generates a text box into which you can type the last name and names that text box Last Name (refer to Figure 3.10 for an example).

Properties in entities expose two particular subproperties, Name and Display Name. The first identifies the control within the application; it cannot contain blank spaces and can be changed, although it is preferable to leave the autogenerated name unchanged. The application infrastructure refers to the control via the Name property (for example, when binding the data to the control). The Display Name property refers to the text you see on the screen, so you can replace this with more meaningful content. Let’s see how to provide the CreateNewContact screen with a more meaningful user interface. Close the application, if it is running, and open the Screen Designer for the CreateNewContact screen by double-clicking it in Solution Explorer. When ready, select the top-level element named Rows Layout, Create New Contact and press F4 to enable the Properties window. Locate the Display Name property and replace its content with Add a New Contact. Figure 3.31 shows this edit.

![Properties screenshot](image)

FIGURE 3.31 Adding a more meaningful description for the Display Name property.

**CHANGING PROPERTIES AT RUNTIME: IMMEDIATE CUSTOMIZATION**

Visual Studio LightSwitch introduces a great feature called **Immediate Customization**, which enables you to rearrange elements in the user interface and set screen properties while the application is running. The feature is described later as you learn about more-complex user interfaces. For now, just know that by clicking the customize screen in the upper-right corner, you can access the customization page on which you can change element properties and arrangement in the user interface. A screen preview shows how changes will appear, and when you save these, the screen layout is automatically updated. The benefit of Immediate Customization is that you do not need to break the application execution every time you want to make a change to the user interface.
After you change display name values, such changes are reflected in the Screen Designer, which shows the new display name for each element. At this point, you can update the user interface for any other screen in the application by following the steps described here. Another step that you can take is to add a description for each property. For example, you might want to indicate to the user how to fill in a particular field, why it is required, or any other useful information. With LightSwitch, you can provide descriptive text inside the Description property in the Properties window. (Refer to Figure 3.31 to understand how to locate it.) Text that you type inside the Description property is shown via a yellow tooltip when you click inside the related field. For example, with regard to the LastName entity’s property, add the following text inside the Description property: **Required. Specifies the contact’s last name.**

Customizations described in this subsection are summarized in Figure 3.29. Before running the application to see how what the user interface now looks like, however, you should replace the application’s name with a more meaningful one.

**Changing the Application’s Name**

When you create a new application, LightSwitch assigns the application a name that is taken from the project name. This makes sense except for the user interface. In fact, considering the current example of the ContactsManager application, you can easily see from the previous figures how the application’s window title reports ContactsManager, but a professional user interface should say Contacts Manager (with a blank space). The application’s name is an application-level setting and can be changed within the Application Designer. So, double-click Properties in Solution Explorer, and on the General tab of the Application Designer, replace the content of the Application Name box with **Contacts Manager**, and then save your changes.

**FIRST CONTACT WITH THE APPLICATION DESIGNER**

The Application Designer is the place where you set application-level properties, including themes, version information, deployment strategy, and security. You will use the Application Designer several times in this book, but each aspect is related to a particular chapter. In this chapter, you just change the application’s name, whereas in Chapter 4 you learn about other features, such as screen navigation. Access control and application types are topics covered, respectively, in Chapters 14, “Debugging LightSwitch Applications,” and 15, “Customizing the IDE.”

At this point, run the application again. You can now see a more professional layout, with the window’s title and descriptions updated with more meaningful information, as shown in Figure 3.32.

**Adding and Removing Data**

You will sometimes need additional data on the entity side and so must add the appropriate control on the screen side to map the new property. LightSwitch makes updating the user interface really easy; you just drag and drop. For example, suppose you want to add a Website property to the Contact entity. To accomplish this, open the Table Designer and
add a new property called WebSite of type String, marking it as not required, and then save your changes. Double-click the CreateNewContact screen in Solution Explorer so that the Screen Designer displays. On the left side of the IDE, you can see that the new WebSite property has been added to the ContactProperty item. Now click the WebSite item and drag it under the Job Role text box, as shown in Figure 3.33.

When you release the mouse button, you can see how LightSwitch adds a new TextBox control, which is data-bound to the WebSite property. This is enough to update the user interface to reflect changes to the data structure.

Repeat the same steps to update the SearchContacts search screen, dragging the WebSite property under the Job Role text box (inside the Data Grid Row element). If you now run the application, you can see how screens have been updated with the new item, which correctly accepts (or shows, in case of the search screen) the specified information, as shown in Figure 3.34.

The important thing to remember here is that you updated the user interface with a simple mouse click, without writing a single line of code to associate the new property with the appropriate text box.
CHAPTER 3  Building Data-Centric Applications

FIGURE 3.33  Dragging the new property onto the designer surface.

FIGURE 3.34  The screen now reflects updates to the entity structure.
Running the Application as a 3-Tier Desktop Client

When you run the application as a 3-tier client from Visual Studio LightSwitch, the ASP.NET development server is used.

*Internet Information Services (IIS)* is actually required only on the web server that you use for deployment, as described further in Chapter 15. As a general rule, you change the application type and deployment topology for your application in the Application Designer. So, in Solution Explorer, double-click Properties. When the Application Designer is available, select the **Application Type** tab on the left. Figure 3.35 shows how things look at this point.

As you can see, LightSwitch shows the list of available application types. You just need to click the desired type and nothing else. Select the **Desktop** client type, and then select the **Host Application Services on an Internet Information Services (IIS) Service application server option** type. When you build the project, Visual Studio LightSwitch generates a 3-tier application in which the middle tier is hosted by IIS and that exposes services responsible for working with data and responsible for communications between the user interface and the data. If you try to run the application, you will see no difference in how the application looks. The reason why is that this application type is, again, a desktop client, even though behind the scenes there is a middle tier working in IIS. For now, this is

---

**FIGURE 3.35** Changing the application type in the Application Designer.
enough. Later in Chapter 15, when discussing deployment and installations, a complete explanation of the IIS requirement is provided.

**Running the Application in the Web Browser**

LightSwitch applications can run inside a web browser such as Microsoft Internet Explorer, Mozilla FireFox, or any other browser that supports Silverlight 4. This is without a doubt a great benefit because you can easily create and deploy applications that end users can use via a web interface. In addition, a 3-tier browser client means that the application is available online only, without the need of a local installation as instead happens for n-tier desktop clients. This is discussed in detail in Chapter 15. For now, let’s see how to run the application in a web browser for testing purposes. Open the Application Designer and go to the Application Type tab shown in Figure 3.32. Now simply select the **Web client application** type. With this simple selection, you switch to a web interface. Press **F5** to run the application again and see the magic happen. The application now runs inside your default web browser and provides the same functionalities that desktop clients offer, as shown in Figure 3.36.

![FIGURE 3.36 Running the application as a 3-tier browser client.](image)

The program is nothing but a complete Silverlight 4 application available online only, running inside a web browser like any other Silverlight application would do, unless you turn it into an out-of-browser application. Before going on, in the Application Designer revert to the default setting, which is a 2-tier desktop client. This is the application type that is used to demonstrate other important LightSwitch built-in features, such as data validation.
Input-Data Validation

Business applications need to validate user input to ensure that entered data is valid, meaningful, and correct with regard to a particular data type. For example, when users enter an email address, the application must validate it to ensure that it is well formed. Other examples are checking that a required field is not empty or checking that a string is shorter than a fixed number of characters or checking that a date is earlier than another one. The main reason for this is that the back-end SQL Server database cannot accept invalid data, and therefore validation is important so that no errors occur when sending data to storage. As a developer, you write validation rules, which are procedures ensuring that data adheres to specific requirements. In other development environments (including Visual Studio 2010), you must write validation rules on your own for every single column in a table. Visual Studio LightSwitch offers a built-in validation mechanism that does most of the work for you, as explained in the following subsections.

Required Fields Validation

Probably the most common requirement in data validation is checking that a required field is neither null nor empty. Visual Studio LightSwitch makes this easy because the generated applications implement a built-in mechanism that automatically checks for empty fields and notify the user about the error. For instance, when you run the Contacts Manager application and you attempt to save changes by leaving the Last Name field empty, the application surrounds the field with a red border and tells you that an error must be fixed before saving changes, as shown in Figure 3.37.

By clicking the error message, you can get detailed information about the issue that caused the error. In this particular case, the user is informed that the Last Name value cannot be null, as shown in Figure 3.38.

MULTIPLE VALIDATION ISSUES

Of course, screens can handle multiple validation issues. If multiple fields do not pass the validation, the screen shows the total number of validation issues at the top of the screen and displays a list of error messages in the details box. This affects not only strings, but any supported data type.

It is important to understand that the error message is not raised by the user interface. Instead, it is raised by the entity, and the user interface is responsible for catching the error, presenting it to the user, and ensuring that validation issues are fixed before saving changes. Entities provide the actual validation mechanism by implementing built-in validation rules.

You can override the default behavior by writing custom validation rules, as described in Chapter 5, “Customizing Data Validation.” For now, focus on the fact that (without writing a single line of code) you have a fully functional, built-in data validation mechanism, which ensures that the user enters correct data without writing a single line of code.
String-Length Validation

Another common requirement in data validation is ensuring that a string is shorter than a fixed number of characters. The default length for the String data type in LightSwitch is 255, but you can make this larger or smaller.

Sometimes this limit is unnecessarily high, so you may decide to reduce the maximum length of a string. For example, the name of a country will never be 255 characters long, so you might want to limit user input for a country name to 50 characters maximum. To accomplish this, you first open the Entity Designer and select the property for which you want to edit validation. Continuing with the Contacts Manager application, select the Country property. Then, open the Properties window and locate the Validation group.
The Validation group in the Properties window is something common to all properties. This group enables you to edit default validation rules for each data type you use in your entity. In addition, it allows you to access the code editor in case you want to write custom validation rules. From now on, it is understood that editing default validation rules is accomplished by locating the Validation group in the Properties window, regardless of the current data type.

To shorten the maximum string length, you just change the value of the Maximum Length text box from 255 to the desired value (to 50 in our example). Figure 3.39 shows what the box looks like after the change. If you previously entered some string longer than the new limit, you get a warning message about data truncation.

![Figure 3.39 Changing a string’s maximum length.](image)

Now run the application and, in the Country field, try to write a string longer than 50 characters. At this point, you will get a validation error informing you that the entered string is longer than allowed.

This ensures that the validation rule is respected, keeping the user interface’s behavior consistent. Figure 3.40 shows this validation scenario.

**Date Validation**

The LightSwitch infrastructure provides default validation mechanisms for the Date type, as well. To understand how this works, open the Table Designer and add a new property named **Anniversary**, of type Date, not required. At this point, open the Properties window and locate the Validation group shown in Figure 3.41.

It is a good idea to provide a suitable range, from the minimum to the maximum value. So, replace the Minimum Value content with **1/1/1920** and the Maximum Value content with **12/31/2000**. This will prevent users from adding people born before January 1, 1920 and after December 31, 2000. Next, following the instructions described earlier in the “Adding and Removing Data” section, open the Screen Designer for the CreateNewContact screen and drag the Anniversary item onto the designer surface. Notice that when you add items of type Date, LightSwitch, by default, wraps them via a DatePicker control. This control displays a calendar that allows easy selection of a date (with just a single click).
FIGURE 3.40  Strings greater than the maximum length are not allowed.

FIGURE 3.41  The Validation group for properties of type Date.
DISPLAYING DATES AND TIME

When mapped to an entity property of type DateTime, the DatePicker control can display either a date or a time interval or both. This is accomplished by selecting the control in the designer and then modifying the appropriate property value in the Properties window. In addition, the DatePicker control can display dates in an extended format (which includes the name of the day and month); just check the Long Date property.

Now run the application and try to specify an out-of-range date in the Anniversary field. Again, the application shows validation errors explaining the details, as shown in Figure 3.42.

![Figure 3.42 Default validation results against items of type Date](image)

FIGURE 3.42 Default validation results against items of type Date.

By taking advantage of the LightSwitch built-in mechanism, you can perform validation against dates without writing a single line of code.
**Number Validation**

The LightSwitch validation mechanism also provides an easy way to validate numbers. This kind of validation is typically used to check whether a number falls within a specified range. This applies to the following types:

- Integer numbers, meaning Short Integer, Integer, and Long Integer data types
- Decimal numbers, meaning the Decimal data type
- Double precision numbers, meaning the Double data type

To understand how it works, consider the Age property, of type Integer, in the Contact entity of the Contacts Manager application.

**DIFFERENCES BETWEEN INTEGER TYPES**

Validation for integer numbers is exactly the same for Int16, Int32, Int64, and Double. The validation mechanism is simply applied by checking whether the number is within the range specified by the MinimumValue and MaximumValue properties. The only difference is in the types themselves, because each has different minimum and maximum values.

Earlier, we decided that only dates between 1920 and 2000 will be accepted, so the person’s age must be within a range of 80 years. If we consider that at the moment when this chapter is being written we are in 2010, according to the dates range, the person cannot be younger than 10 and cannot be older than 90. With that said, open the Entity Designer for the Contact entity, and then select the Age property. Then open the Properties window. Locate the Validation group and replace the content of the Minimum Value and Maximum Value (empty by default) fields, respectively, with 10 and 90.

**USING COMPUTED PROPERTIES**

This discussion requires manual calculations to fit the age of a person with a date range. Actually, LightSwitch provides a more convenient way to automate calculations based on other properties in the entity, known as computed properties (discussed in Chapter 4). The current example is just to demonstrate validation against numbers.

If you now run the application and try to enter a value for the Age field that does not fall within the expected range, you get a validation error, as shown in Figure 3.43.

**Notes About Decimal Numbers**

Numbers of type Decimal provide two additional properties for validation: Precision and Scale. The first one represents the maximum number of digits for the value in the entire field, including digits that are both on the left and on the right of the decimal point. Scale allows you to specify the number of digits to the right of the decimal point.
In this chapter, you learned that Visual Studio LightSwitch introduces the concept of business data types. You also learned what business data types are about and why they are so useful. Continuing the discussion about data validation, business data types provide default validation mechanisms specific to their role. For instance, think of email addresses. These are not simply text; they are something requiring specific validation, such as the presence of a domain name or of the @ (at) symbol. LightSwitch provides a sophisticated validation mechanism that does the work for you, for all business data types. This section explains the default validation of the various business types.

**SUPPORT FOR CUSTOM VALIDATION RULES**

As with primitive types described earlier in this chapter, the validation mechanism for business data types can be personalized by writing custom validation rules. For more information, see Chapter 5.

**Validating Email Addresses**

The Email Address data type has a built-in validation mechanism that checks whether the supplied email address is well formed. Therefore, if the user enters an invalid email address, the application throws a notification reporting the error and prevents invalid
changes from being saved. Figure 3.44 shows the validation error the user receives if a pound symbol (#) is entered rather than the required @.

![Image showing validation error](image)

**FIGURE 3.44** Default validation on **Email Address** data types.

After you fix the error, the validation test passes and you can save your changes.

**Setting the Default Domain**

The validation mechanism for the **Email Address** data type works against three properties: **IsRequired** and **MaximumLength**, which you already encountered when studying other primitive types; and **Require Email Domain**. When enabled, this last one requires the user to supply a valid domain name in the email address. For example, in my email address, the domain is @visual-basic.it. This can be quite annoying if you want to build a list of contacts from the same group of people, such as your colleagues, because in such a circumstance, all of your contacts’ email addresses have the same domain. Fortunately, the **Email Address** data type also provides the possibility of specifying a default domain that is automatically added to each email address. To provide a default domain, follow these steps:

1. In the Entity Designer for the **Contact** entity, select the **Email** property and open the Properties window.
2. Locate the Validation group and uncheck the **Require Email Domain** property.
3. Locate the General group and, inside the Default Email Domain field, type the desired email domain. If you do not add the @ symbol at the beginning, LightSwitch adds it for you.

When you run the application next, you will notice that each time you add an email address, the specified domain is automatically added for you, saving a lot of time.

Validating Phone Numbers

Validation for properties of type Phone Number works as it does for strings, in that you can just specify whether the property is mandatory (IsRequired) and the length of the string representing the phone number (255 characters by default). More than for validation, the Phone Number data type is interesting for its customization and formatting methods, described earlier in this chapter.

Validating Images

The Image data type has just one validation property: IsRequired. This is because this type works in conjunction with controls that automatically determine whether an image is valid and can be accepted by properties of type Image.

Validating Money

The Money business data type is basically an evolution of the Decimal primitive type. In fact, their behavior is pretty identical; the difference is that Money provides the currency symbol and localized information according to the system’s regional settings. So, default validation for this type checks for the same properties as in Decimal: MinimumValue, MaximumValue, Precision, and Scale.

Summary

Visual Studio LightSwitch wants developers to be able to develop business applications quickly and easily. This kind of application is all about data, so in this chapter, you learned how LightSwitch organizes data in tables, entities, and entity collections. Then you learned how entities can handle specific data types, such as strings, integer numbers, dates, email addresses, and so on. This chapter also explained the main steps you follow in the development process:

- Create a new Visual Basic or Visual C# project.
- Design data sources via the Table Designer, by providing properties and setting data types for each property.
- Add screens from a set of common templates, understanding how LightSwitch automatically binds data to the user interface.
- Test the application by running it as a 2-tier or 3-tier client, both on the desktop and in the web browser.
- Validate data and customize the default validation behaviors.
An overview of the most common controls and of the application architecture was provided to give you the basics about topics that are discussed in more detail later in this book. This chapter provided the basics of the LightSwitch development for a single-table database and a user interface that has no relationships with other screens. But real-world applications are more complex, both on the data side and on the user interface side. This is what we cover in the next chapter.
Index

A

Access, 4-6
  benefits, 4
  data entry form, 4
  drawbacks, 6
adding, phone numbers, 64-65
adding relationships, 105-112
administrators, specifying, 314-319
aggregating data, 236
  data sources, 236-256
analysis, execution flow, IntelliTrace, 512-515
Application Designer, 34-35
  General Properties, 345
Application icon property (Application Designer), 345
Application name property (Application Designer), 345
Application Server Configuration screen (Publish Application Wizard), 353
Application version property (Application Designer), 345
Applications, 443. See also projects
  aggregating data into, 250-256
  architecture, 444-445
  business, 1-2
  custom controls, 561-562
    building, 562-563
    creating with Visual Studio 2010, 563-573
  customizing, command bars, 216-228
data-centric, 43, 93-94
  adding entity properties, 46-50
  creating, 43-44
creating data sources, 45-52
data entry screens, 55-58
data validation, 87-89
default validation of business types, 91-93
exporting data to Excel, 77-78
implementing screens, 52-60
input-data validation, 85-90
number validation, 90
required fields validation, 85
running as 3-tier desktop client, 83-84
running in web browsers, 84
screen customization, 78-84
search screens, 58-60
string-length validation, 86-87
testing, 60-84
data storage, 444
debugging, 36, 312, 517-527
  analyzing threads, 534
  breakpoints, 521-522, 527-532
  code, 536-541
data tips, 521-522
debugger visualizers, 535-536
displaying variable values, 530
evaluating expressions, 531-532
  method calls, 532-533
runtime errors, 524-526
  steps, 522-524
trace points, 527-532
Watch windows, 533-534
deploying, 312, 339-344
  MSDeploy, 343-344
one-click deployment, 340-341
preparation, 344-352
runtime settings, 349-350
three-tier applications, 373-392
two-tier applications, 352-373
Windows Azure, 392-407
extending, SharePoint 2010, 280-290
life cycles, managing, 480-506
localizing, 350-352
logos, specifying, 345-348
naming, 80, 345-348
out-of-browser, 60
publishing
  Forms authentication, 330-334
  Windows authentication, 314-319
running, 116-118
SQL Server databases, creating on, 239-249
starting as desktop clients, 61-63
styling, 348-349
technologies, 444-445
testing, credentials, 327-329, 334-335
three-tier, 341-344
  publishing, 377-379
tiers
  data storage, 445-452
  logic, 444, 452-462
  presentation, 445, 462-469
two-tier, 341-344
  publishing, 368-373
WCF RIA Services, calling from, 720-726
applying filters
  data level, 181-205
  screen level, 206-208
appointments, Outlook, creating, 586
architecture, applications, 444-445
arranging windows, 37-38
asynchrony, 178
attributes, debuggers, 538-541
authentication, 291-294, 338
  Forms authentication, implementing, 329-335
mechanisms, RIA services, 717
strategies, choosing, 295
Windows authentication, implementing, 294-329
authorization, 291-292
permissions
  entity logic, 299-307
  logic, 298-299
  user interface logic, 307-312
settings permissions, 297-298
Autogenerated Code for CreateNewCustomer listing (12.1), 477
Azure
  configuring
    role instances, 407
  services, 398-402
  SQL, 402-404
  connecting to, 256-273, 394-398
  deploying applications to, 392-407

B
Binary data type, 47
binding
  imported tables, to screens, 253-256
  user controls to screens, 569-572
Bing Maps
  adding to screens, 576-578
  developer keys, claiming, 574
  integrating, 573-578
  making available to LightSwitch, 575
  updating entities, 576
blogs, 766, 768
Boolean data type, 47
breakpoints, 527-532
  Breakpoints window, 527-528
  code, 521-522
  labels, editing, 528
  Breakpoints window, 527-528
Building a Composite Control for Data Visualization listing (16.1), 566
Building a Custom Code Snippet listing (15.2), 556-557
Building a Silverlight Reporting Control listing (17.1), 610-612
building custom controls, 562-563
built-in query events, handling, 424-427
built-in validation rules, 152-153
business applications, 1-2
business data types, 49-50
business types, 228
  creating, 693-705
  default validation, 91-93
business-oriented user interfaces, 115-146
Button control, 54
button methods, screen events, 431-434
buttons
  built-in, 216
    CanExecute method, 312
    command bars, adding to, 217-218
    custom, adding and managing, 219-222

C
C# 4 Unleashed, 22
calendars, SharePoint 2010, configuring, 275-280
Call Stack window, method calls, 532-533
calling WCF RIA Services, 720-726
  stored procedures, 726-745
CanExecute method, 312

How can we make this index more useful? Email us at indexes@samspublishing.com
change sets, 459
chart controls, implementing, 565-569
CheckBox control, 55
choice lists, defining, entities, 100-103
Chowdhury, Kunal, 768
class coupling index (Code Metrics), 506
class library, Office Integration Extension, 582-583
classes
   Debug, 536-538
   RIA services, 710
ClickOnce, 340-341
client operating systems, IIS (Internet Information Services), 342
Client project, 475-477
client validation, 154-167
ClientGenerated project, 475
client-side projects, 474-478
   Client, 475-477
   ClientGenerated, 475
   compiled files, 477-478
code
   breakpoints, 521-522, 527-532
   debugging, 536-541
   entities, 409-429
      data objects, 410-413
   events
      data-related operations, 416
         handling, 409, 429-438
   queries, customizing, 195-205
   runtime errors, 524-526
   screens, adding to, 678-685
   trace points, 527-532
   unit testing, 508-512
Code Editor, 35
code listings, 696
   Autogenerated Code for CreateNewCustomer, 477
Building a Composite Control for Data Visualization, 566
Building a Custom Code Snippet, 556-557
Building a Silverlight Reporting Control, 610-612
Creating a Custom Screen Template, 680-685
Creating a User Control for the WebAddress Type, 701
Definition File for the Custom Theme, 625-626
Editing the Control’s Definition File, 702
Examining an Existing Code Snippet, 555
Implementing a Business Object to Represent a Single Feed, 712
Implementing a Domain Service Class, 715
Implementing a Metadata Class for the Domain Service, 718
Implementing a Validator Class for the WebAddress Type, 697
Implementing Properties in the Control Definition File, 661-663
Implementing Security Methods for Multiple Entities, 304-305
Implementing Security Methods on Custom Queries, 306
Implementing the Control’s UI, 663-664
Implementing the Validator Factory Class, 699
Implementing the Viewer Control, 702
Modifying the Code for Debugging Purposes, 519
Representing the New Business Type in a .NET Manner, 696
Sending Email via Outlook Automation, 224-225
Setting Security Permissions at the Screen Level, 308-311
WebAddress.lsm Definition File, 695
Code Metrics, 506-508
   Indexes, 506
code samples, 767
Code Snippet Manager, 553-554

code snippets
    creating custom, 554-559
    using, 551-554

CodeProject, 768

collapsing screens, navigation panel, 187

collection methods, screen events, 437-438

collections, entity, validation, 169-170

Columns Layout control, 138

COMM interoperability, 16, 222-228

command bars
    buttons
        adding to, 217-218
        managing custom, 219-222
        customizing, 216-228

command panel, styling, 641-644

Command window, evaluating expressions, 531-532

community resources, 767-768

company logo, displaying, 653-654

comparison operators, 182-183

compiled files, 477-478

complex data sources, designing, 97-115

Component One, OLAP, 606-608

composite control for data visualization, building, 566

compound properties, 112-115

computed properties, controls, 122

configuration
    Azure
        role instances, 407
        services, 398-402
        SQL, 402-404
    IntelliTrace, 513-514
    LightSwitch, 759
    web servers, 374-376

connection strings, handling, 749-751

connections
    SQL Server, 256-273
        databases, 236-256
    TFS (Team Foundation Server), 481-482

ContactsManager application, 93-94
    creating, 43-44
    data sources, creating, 45-52
    data validation, 87-89
    default validation of business types, 91-93
    entity properties, adding, 46-50
    exporting data to Excel, 77-78
    input-data validation, 85-90
    number validation, 90
    running as 3-tier desktop client, 83-84
    running in web browsers, 84
    screen customization, 78-84

screens
    data entry, 55-58
    implementing, 52-60
    search, 58-60
    string-length validation, 86-87
    testing, development machines, 60-84

close tree, screens, 466-468

controls, 21, 53-55, 619-620
    chart, implementing, 565-569
    computed properties, 122
    custom, 561-562
        building, 562-563
        sharing, 656-670
    custom controls, 228
    dependency properties, implementing, 664-667
    designing, 700-703
    editing, 76-77
    extensions, creating, 657-670
    icons, replacing, 658
    making programmable, 667

How can we make this index more useful? Email us at indexes@samspublishing.com
Modal Window, 138
MSDN documentation, 669-670
Picture and Text, 138
printing, custom, 609-617
reporting, custom, 609-617
Rows Layout, 138
Screen Navigation, 133-138
shells, restyling, 655-656
string resources, adding, 658-659
Table Layout, 138
Tabs Layout, 138
testing, 667-669
Text and Picture, 138
types, 657
user interfaces, designing, 663-664
viewer, creating, 701-703
CreateNewCustomer, 477
Creating a Custom Screen Template listing (18.4), 680-685
Creating a User Control for the WebAddress Type listing (19.5), 701
credentials
    applications, testing, 334-335
testing applications, 327-329
Culture property (Application Designer), 345
currency codes, 103
custom buttons, adding and managing, 219-222
custom code snippets, creating, 554-559
custom controls, 228, 561-562
    Bing Maps, integrating, 573-578
    building, 562-563
    creating with Visual Studio 2010, 563-573
    printing, 609-617
    reporting, 609-617
    sharing, 656-670
testing, 667-669
custom data sources
    creating, 705-745
    sharing, 745-751
testing, 746-749
custom query events, handling, 429-430
custom rules, validation, writing, 153-174
custom shells
    creating, 636-656
    extensibility projects, creating for, 638-639
testing, 654-655
custom themes, testing, 633-634
Customer entity, 97-98
customization
    applications, command bars, 216-228
data validation, 149
IDE (integrated development environment), 543
toolbars, 544-545
queries, code, 195-205
screens, 78-81, 138-146
cyclomatic complexity index (Code Metrics), 506

D
data
    asynchrony, 178
    editing, 118-121
    filtering, 177, 179-208
    paging, 178
    querying, 177
        based on other queries, 212-214
    sorting, 177
        data level, 208-211
        logic, 208-211
        screen level, 211
data access tier, applications, 445-452
data aggregation, data sources, 236-256
data connections, 16
data entry forms, Access, 4
data entry screens, creating, 55-58, 116
data level
  data, sorting, 208-211
  filters, applying, 181-205
data objects, 410-413
data paging, 16
data providers
  logic tier, 461
  SharePoint 2010, 448
  SQL Azure, 445-446
  SQL Server, 445-446
data service clients, presentation tier, 468-469
data services, logic tier, 452-453
Data Source Methods, data source events, handling, 423-424
data sources, 229
  aggregating data from, 235-256
  creating, 705-745
  custom
    sharing, 745-751
    testing, 746-749
  data-centric applications, creating, 45-52
  designing, 101-115
  events, handling, 423-424
data storage, 52
  applications, 444
data storage tier, applications, 445-452
data tips, 521-522
data types, 47-49
  business, 49-50
  definition, implementing, 694-696
  Integer, 49-50
  mapping, 49, 448-452
  Money, 103-105
  validating, 160-165
data validation, 16, 175
  built-in rules, 152-153
  custom rules, writing, 153-174
  customizing, 149
  data-centric applications, 87-89
  entity collections, 169-170
  implementing, 146-148, 696-700
  model, 150-153
  rule types, 151
data workspace, presentation tier, 468
Database Connections screen (Publish Application Wizard), 356-357
database tools, 20-21
databases
  aggregating data from, 250-256
  intrinsic, 446-447
  membership, 447-448
  SQL Azure, connecting to, 256-273
  SQL Server
    connecting to, 236-256
    creating applications on, 239-249
data-binding, 16
data-centric applications, 43, 93-94
  creating, 43-44
  data sources, creating, 45-52
  data validation, 87-89
  default validation of business types, 91-93
  entity properties, adding, 46-50
  Excel, exporting data to, 77-78
  input-data validation, 85-90
  number validation, 90
  required fields validation, 85
  running as 3-tier desktop client, 83-84
  running in web browsers, 84
  screen customization, 78-84
screens
  data entry, 55-58
  implementing, 52-60
  search, 58-60
  string-length validation, 86-87
  testing, development machines, 60-84
DataGrid control, 55
DataGridRow control, 55
data-related operations, events, 416
Date data type, 47
Date Picker control, 54
Date Viewer control, 54
dates, displaying, 89
DateTime data type, 47
DateTimePicker control, 54
DateTimeViewer control, 54
de Smet, Bart, 22
Debug class, 536-538
DebuggerBrowsable attribute, 539
DebuggerDisplay attribute, 539
DebuggerHidden attribute, 540
debuggers
  attributes, 538-541
  visualizers, 535-536
DebuggerStepThrough attribute, 540
DebuggerTypeProxy attribute, 541
debugging, 531-532
  applications, 36, 312, 517-527
    analyzing threads, 534
    breakpoints, 521-522, 527-532
    data tips, 521-522
    debugger visualizers, 535-536
    displaying variable values, 530
    Error List tool window, 520
    evaluating expressions, 531-532
    method calls, 532-533
    runtime errors, 524-526
  steps, 522-524
    trace points, 527-532
    Watch windows, 533-534
code, 536-541
Decimal data type, 47
decimal numbers, 90
default queries, 180
default validation of business types,
data-centric applications, 91-93
defining choice lists, entities, 100-103
definition, data types, implementing, 694-696
Definition File for the Custom Theme listing (18.1), 625-626
deleting, entities, 422
dependency calculation, logic tier, 457
dependency properties, implementing, 664-667
deploying, extensions, 751-758
deploying applications, 312, 339-344
  MSDeploy, 343-344
  one-click deployment, 340-341
  preparation, 344-352
  runtime settings, 349-350
  three-tier applications, 373-392
  two-tier applications, 352-373
    Publish Application Wizard, 353-373
    Windows Azure, 392-407
deployment manifest, preparing, 752
depth of inheritance index (Code Metrics), 506
Designer, 29-30
designing
  controls, 700-703
  screen templates, 672-673
desktop clients, starting applications as, 61-63
details lists, creating, 129
Details screen, master-details relationships, handling, 121-133
details screens, 95-96
Developer Center, 21
Developer Express, XtraReports, 600-605
developer keys, Bing Maps, claiming, 574
developers, 3
development environments, 19
display names, customizing, 79, 185
displaying reports, user interfaces, 612-613
Document Toolkit for LightSwitch, 769
Double data type, 48
downloading extensions, 229-232
drawbacks, Access, 6

E

Edit and Continue feature, 526-527
editable grids, 76, 118-121
editing
  breakpoint labels, 528
data, 118-121
relationships, 109-110
Screen Navigation control, 133-138
themes
  Expression Blend 4, 630-631
  Visual Studio 2010, 628-629
Editing the Control's Definition File listing
  (19.6), 702
Editor control, creating, 700-701
elevation, permissions, server code, 336-337
e-mail, sending, Outlook, 224-225, 583-585
Email Address data type, 48
Email Address Editor control, 54
Email Address Viewer data type, 54
e-mail addresses, validating, 91-93
entities
  adding properties, 46-50
  building, 50-52
  choice lists, defining, 100-103
  code, 409-429
  data objects, 410-413
Customer, 97-98
deleting, 422
intrinsic database, adding to, 250-253
Invoice, 102
managing, presentation tier, 469
OrderDetail, 99
OrderHeader, 98
permissions, logic, 299-307
Product, 100
versus tables, 46
updating, Bing Maps, 576
validation, single entity, 167-169
entity collections, data validation, 169-170
Entity Designer, 30
entity properties, validating, 154-156
Error List tool window, 520
errors, runtime, 524-526
event method handlers, 414
events
code
  data-related operations, 416
  handling, 429-438
handling, 409
entities, 409-429
property-related events, handling, 427-429
queries, 205
query events, handling, 424-427, 429-430
screen events
  button methods, 431-434
  collection methods, 437-438
  general methods, 434-436
  handling, 430-438
Examining an Existing Code Snippet listing
  (15.1), 555

How can we make this index more useful? Email us at indexes@samspublishing.com
Excel

exporting data to, 77-78, 593-594
importing data from, 594-597
exceptions, save pipeline, 459-460
execution flow, analyzing, IntelliTrace, 512-515
exporting
data to Excel, 77-78, 593-594
data to PDF documents, 586-592
data to Word documents, 586-592
user settings, 547-548
Expression Blend 4, themes, editing, 630-631
evaluating, Command window, 531-532

extending applications, SharePoint 2010, 280-290
extensibility, MEF (Managed Extensibility Framework), 620-621
Extensibility Center, 767
extensibility projects
creating, 621-624
custom shells, creating for, 638-639
themes, adding to, 625-628
Extensible Application Markup Language (XAML), 620
extensions, 228-229
business types, 228
controls, creating, 657-670
custom controls, 228
data sources, 229
deploying, 693
deploying to others, 751-758
downloading and installing, 229-232
NetAdvantage, 605-606
OLAP, 606-608
Office Integration Extension, 582-597
properties, setting, 624
reporting, 599-600
NetAdvantage, 605-606
OLAP, 606-608
Telerik, 606-608
XtraReports, 600-605
screen templates, 228
shells, 228
testing, 703-705
themes, 229
third-party, 769-770
updates, releasing, 757-758
using, 232-233

F

files, compiled, 477-478
Filter Designer, 181-184
filtering data, 70-72, 177, 179-208
filters
applying
data level, 181-205
screen level, 206-208
comparison operators, 182-183
Filter Designer, 181-184
group filters, adding, 190-192
query parameters, 193-200
formats, phone numbers, customizing, 65-68
formatting, phone numbers, 64-65
Forms authentication, 338
implementing, 329-335
forums, LightSwitch, 766
G
general methods, screen events, 434-436
General Properties, Application Designer, 345
gradient stops, editing, 628
group filters, adding, 190-192
Guide data type, 48

H
handling
  connection strings, 749-751
events, 409, 430-438
  screen, 430-438
master-details relationships, Details
  screen, 121-133
relationships, 110-112
Haugen, Nicole, 766
Help Viewer, 38-39
helper code, unit testing, 508-512
hosting service, presentation tier, 463
"How-do-I" videos, 766

I
icons, controls, replacing, 658
IDE (integrated development environment), 23-25, 43, 543, 559
  Application Designer, 34-35
  Code Editor, 35
code snippets
  creating custom, 554-559
  using, 551-554
creating new projects, 26-29
customizing, 543
  command bars, 216-228
Designer, 29-30
Entity Designer, 30
extensions, 228-229
  business types, 228
custom controls, 228
data sources, 229
downloading and installing, 229-232
screen templates, 228
shells, 228
themes, 229
using, 232-233
Help Viewer, 38-39
Properties window, 32
Query designer, 33-34
Screen Designer, 30-31
Solution Explorer, 27-29
Start Page, 25-26
toolbars, customizing, 544-545
user settings, managing, 546-550
windows, managing and arranging, 37-38
IIS (Internet Information Services), 20
  client operating systems, 342
Image data type, 48
Image Editor control, 54
Image Viewer control, 54
images, validating, 93
Implementing a Business Object to Represent a
  Single Feed listing (19.8), 712
Implementing a Domain Service Class listing
  (19.9), 715
Implementing a Metadata Class for the Domain
  Service listing (19.10), 718
Implementing a Validator Class for the
  WebAddress Type listing (19.3), 697
Implementing Properties in the Control
  Definition File listing (18.2), 661-663
Implementing Security Methods for Multiple
  Entities listing (9.1), 304-305

How can we make this index more useful? Email us at indexes@samspublishing.com
Implementing Security Methods on Custom Queries listing (9.2), 306
Implementing the Control’s UI listing (18.3), 663-664
Implementing the Validator Factory Class listing (19.4), 699
Implementing the Viewer Control listing (19.7), 702
imported tables, screens, binding to, 253-256
importing
  MSDeploy packages, IIS (Internet Information Services), 385-392
  user settings, 548-550
importing data from Excel, 594-597
indexes, Code Metrics, 506
Infragistics, NetAdvantage, 605-606
Infragistics NetAdvantage Light Edition, 770
INotifyDataErrorInfo, 457-458
input-data validation, data-centric applications, 85-90
installation
  extensions, 229-232
  LightSwitch, 19-21, 759-761
  Northwind database, 237-238
Integer data type, 48-50
integer types, differences, 90
integrated development environment (IDE). See IDE (integrated development environment)
integration, Bing Maps, 573-578
IntelliTrace
  configuring, 513-514
  execution flow, analyzing, 512-515
  log files, 515
interfaces, business-oriented, 115-146
interoperability (COM), 222-228
intrinsic database, 446-447
  entities, adding to, 250-253
invalid data, saving, 422-423
Invoice entity, 102
IScreenTemplate properties, implementing, 673-675
J
Jennings, Roger, 768
L
Label control, 54
labels, breakpoints, editing, 528
Language Integrated Query (LINQ), 171
launching screens programmatically, 438-442
life cycles, applications, managing, 480-506
LightSwitch, 13-19, 22
  configuring, 759
  editions, 18
  installing, 19-21, 759-761
LightSwitch Developer Center, 765
LightSwitch Team, 766
limitations, Visual Basic 6, 7-8
lines of code index (Code Metrics), 506
LINQ (Language Integrated Query), 171
List control, 55
listings
  Autogenerated Code for
    CreateNewCustomer, 477
  Building a Composite Control for Data Visualization, 566
  Building a Custom Code Snippet, 556-557
  Building a Silverlight Reporting Control, 610-612
  Creating a Custom Screen Template, 680-685
  Creating a User Control for the WebAddress Type, 701
Definition File for the Custom Theme, 625-626
Editing the Control’s Definition File, 702
Examining an Existing Code Snippet, 555
Implementing a Business Object to Represent a Single Feed, 712
Implementing a Domain Service Class, 715
Implementing a Metadata Class for the Domain Service, 718
Implementing a Validator Class for the WebAddress Type, 697
Implementing Properties in the Control Definition File, 661-663
Implementing Security Methods for Multiple Entities, 304-305
Implementing Security Methods on Custom Queries, 306
Implementing the Control’s UI, 663-664
Implementing the Validator Factory Class, 699
Implementing the Viewer Control, 702
Modifying the Code for Debugging Purposes, 519
Representing the New Business Type in a .NET Manner, 696
Sending Email via Outlook Automation, 224-225
Setting Security Permissions at the Screen Level, 308-311
WebAddress.Isml Definition File, 695
lists, SharePoint 2010, 273-290
literal properties, 558
localizing, applications, 350-352
Locals window, variable values, displaying, 530
log files, IntelliTrace, 515
logic
data sorting, 208-211
permissions, writing, 298-299
logic tier, applications, 444, 452-462
change sets, 459
data providers, 461
data services, 452-453
dependency calculation, 457
queries, 453-454
static spans, 454-455
transaction management, 459-460
validation framework, 456
Logo image property (Application Designer), 345
logos
applications, specifying, 345-348
displaying, 653-654
LongInteger data type, 48
maintainability index (Code Metrics), 506
Managed Extensibility Framework (MEF), 620-621
managing
custom buttons, 219-222
offline documentation, 762-763
user settings, 546-550
many-to-many relationships, 105-108
mapping
data types, 49, 448-452
stored procedures, .NET methods, 727-736
Massi, Beth, 766
master-details relationships
handling, Details screen, 121-133
validation, 147-148, 170-172
MEF (Managed Extensibility Framework), 620-621
membership database, 447-448
method calls, Call Stack window, 532-533
methods
button, screen events, 431-434
CanExecute, 312
How can we make this index more useful? Email us at indexes@samspublishing.com
collection, screens, 437-438

general, screen events, 434-436

Metro theme, 621

middle-tier projects, 473-474

Modal Window control, 138

Modifying the Code for Debugging Purposes listing (14.1), 519

money, validating, 93

Money data type, 48, 103-105

Money Editor control, 55

Money View control, 55

MSDeploy, 343-344

packages, creating and importing, 385-392

MSDN, resources, 765

MSDN documentation, controls, 669-670

multiple validation issues, 85

MyVBProf.com, video training, 768

O

Oakleaf Systems, 768

objects, data objects, 410-413

OData, endpoints, exposing, 713

Office, integration, 78

Office Integration Extension, 582-597

class library, 582-583

Excel

exporting data to, 593-594

importing data from, 594-597

Outlook

creating appointments, 586

sending email, 583-585

PDF documents, exporting data to, 586-592

Word documents, exporting data to, 586-592

offline documentation, managing, 762-763

OLAP, 606-608

one-click deployment, applications, 340-341

one-to-many relationships, 105

one-to-one relationships, 105-108

operating systems, requirements, 19

operators, comparison, 182-183

OrderDetail entity, 99

OrderHeader entity, 98

Other Connections screen (Publish Application Wizard), 360-363

out-of-browser applications, 60

out-of-browser functionality, 16

Outlook

appointments, creating, 586

automation, sending email via, 224-225

sending email, 583-585

N

namespaces, RIA services, 710

naming, applications, 80, 345-348

navigation, screens, rearranging, 185

Navigation area, styling, 650-653

navigation panel, collapsing, 187

.NET data types, 449-450

.NET Framework, 21-22

validation rules, implementing, 172-174

WCF RIA Services, 709-711

.NET methods, stored procedures, mapping, 727-736

NetAdvantage, 605-606

Northwind database, installing, 237-238

null checks, 162

number validation, data-centric applications, 90
Paging data, 70, 178
Patterson, Paul, 768
PDF documents, exporting data to, 586-592
permissions
  elevation, server code, 336-337
  logic
    entities, 299-307
    user interfaces, 307-312
    writing, 298-299
settings, 297-298
user roles, 320-323
Phone Number data type, 48
Phone Number Editor data type, 54
Phone numbers
  adding and formatting, 64-65
  customizing formats, 65-68
  validating, 93
Picture and Text control, 138
preparation, application deployment, 344-352
Prerequisites screen (Publish Application Wizard), 357-360
presentation tier, applications, 445, 462-469
  data service clients, 468-469
  data workspace, 468
  hosting service, 463
  managing entities, 469
  screens, 465-468
  shell, 463-465
  theming service, 465
printing
  APIs, 613-617
  custom controls, 609-617
  implementing, 581-582
  Office Integration Extension, 582-597
printing APIs, 613-617
Product entity, 100
programmatically launching screens, 438-442
programmable controls, creating, 667
programming, 21-22
projects. See also applications
  automating builds, 499-506
  client-side, 474-478
  Client, 475-477
  ClientGenerated, 475
  compiled files, 477-478
  creating, 26-29, 96-97
  dissecting, 469-478
  middle-tier, 473-474
  server-side, 471-473
    Server, 472-473
    ServerGenerated, 472
  solutions, 470-471
  test projects, creating, 509-510
  version control, 499
properties
  changing, runtime, 79-80
  compound, 112-115
  entity
    adding, 46-50
    validation, 154-167
  extensions, setting, 624
  literal, 558
  summary, 73-76
  testing, 76
Properties window, 32
property-related events, handling, 427-429
Publish Application Wizard
  Application Server Configuration screen, 353
  Database Connections screen, 356-357
  Other Connections screen, 360-363
  Prerequisites screen, 357-360
  Publish Output screen, 353

How can we make this index more useful? Email us at indexes@samspublishing.com
Publish Summary screen, 366-368
Specify a Certificate screen, 363-366
two-tier applications, deploying, 353-373
Publish Output screen (Publish Application Wizard), 353
Publish Summary screen (Publish Application Wizard), 366-368
publishing
applications
Forms authentication, 330-334
Windows authentication, 314-319
three-tier applications, 377-379
MSDeploy packages, 385-392
to web servers, 379-383
two-tier applications, 368-373

Q
queries
basing on other queries, 212-214
code, customizing, 195-205
default, 180
events, 205
logic tier, 453-454
query parameters, 193-200
Query designer, 33-34
Query Designer, 181-182
query events, handling, 424-427
custom, 429-430
query parameters, 193-200
querying, data, 177-179

R
RAD (Rapid Application Development)
environment, 43
refreshing, search results, 72
relationships, 95-96
adding, 105-112
editing, 109-110
handling, 110-112
many-to-many, 105-108
master-details
handling, 121-133
validation, 147-148, 170-172
one-to-many, 105
one-to-one, 757-758
releasing, extension updates, 757-758
renaming, applications, 80
reporting
custom controls, 609-617
extensions, 599-600
XtraReports, 600-605
implementing, 581-582
Office Integration Extension, 582-597
SQL Server Reporting Services, 597-599
reports
creating as user control, 609-612
displaying, user interface, 612-613
Representing the New Business Type in a .NET Manner listing (19.2), 696
required fields validation, data-centric applications, 85
resources
community, 767-768
MSDN, 765-767
third-party extensions, 769-770
restyling controls, shells, 655-656
RIA services
authentication mechanisms, 717
classes, 710
namespaces, 710
Ribbon Bar, buttons, adding to, 217-218
role instances, Azure, configuring, 407
roles, users
  creating, 312-329
  permissions, 320-323
root container, styling, 641
Rows Layout control, 138
RSSBus data providers, 769
rule types, validation, 151
rules, validation, 456-457
  writing custom, 153-174
running applications, 116-118
runtime, changing properties, 79-80
runtime components, 16
runtime errors, 524-526
runtime settings, application deployment, 349-350

S
Sampson, Matt, 766
save pipeline, 424, 458-459
  exceptions, 459-460
saving, invalid data, 422-423
Screen Command Bar control, 54
Screen Content area, styling, 644-650
screen content trees, generating, 675-678
Screen Designer, 30-31
screen events
  button methods, 431-434
  collection methods, 437-438
  general methods, 434-436
  handling, 430-438
screen level
  data, sorting, 211
  filters, applying, 206-208
Screen Navigation control, editing, 133-138

screen templates, 228
  creating, 670-691
  designing, 672-673
  details, entity and items, 687-690
  testing, 685-686
screens, 619-620
  adding code to, 678-685
  Bing Maps, adding to, 576-578
  customization, 78-81, 138-146
  data entry, creating, 55-58, 116
  data-centric applications, implementing, 52-60
  display names, customizing, 185
  imported tables, binding to, 253-256
  launching programmatically, 438-442
  navigation panel, collapsing, 187
  predefined templates, 31
  presentation tier, 465-468
  rearranging navigation, 185
  screen templates, creating, 670-691
  search, creating, 58-60, 116-118
  templates
    adding local screen members, 690-691
    creating, 678-685
    designing, 672-673
    details, 687-690
    IScreenTemplate properties, 673-675
    testing, 685-686
    user controls, binding to, 569-572
search results, refreshing, 72
search screens
  creating, 58-60, 116-118
  editing data from, 72
searching, data, 70-72

How can we make this index more useful? Email us at indexes@samspublishing.com
security

authentication, 291-294
  Forms authentication, 329-335
  strategies, 295
  Windows authentication, 294-329
authorization, 291-292
  settings permissions, 297-298
permissions
  entity logic, 299-307
  server code elevation, 336-337
  user interface logic, 307-312

sending email
  automation, 224-225
  Outlook, 583-585

Sending Email via Outlook Automation listing (7.1), 224-225
server code, permission elevation, 336-337
server components, 19-20
Server project, 472-473
ServerGenerated projects, 472
server-side projects, 471-473
  Server, 472-473
  ServerGenerated, 472
services, Windows Azure, configuring, 398-402
setting properties, extensions, 624

Setting Security Permissions at the Screen Level listing (9.3), 308-311
settings permissions, 297-298
SharePoint 2010, 20, 448
  calendars, configuring, 275-280
  extending applications, 280-290
  lists, 273-290
sharing
  custom controls, 656-670
  custom data sources, 745-751
Shell property (Application Designer), 345
shell UI, presentation tier, 463-465

shells, 228, 619-620
  command panel, styling, 641-644
  company logo, displaying, 653-654
  controls, restyling, 655-656
  creating, 636-656
  extensibility projects, creating for, 638-639
  Navigation area, styling, 650-653
  root container, styling, 641
  Screen Content area, styling, 644-650
  testing, 654-655
  User Information area, styling, 653
  view models, 640-641

ShortInteger data type, 48
Silverlight, Telerik, 608
single entity, validation, 167-169
snippets (code)
  creating custom, 554-559
  using, 551-554
software, 1
software developers, 3
Solution Explorer, 27-29
solutions, projects, 470-471

sorting
data
  data level, 208-211
  logic, 208-211
  screen level, 211
data-centric applications, 177

source control, team projects, submitting to, 485-492

Specify a Certificate screen (Publish Application Wizard), 363-366

spreadsheets (Excel)
  exporting data to, 593-594
  importing data from, 594-597

SQL (Structured Query Language), 171
SQL Azure, 21, 445-446
  configuring, 402-404
  connecting to, 256-273
SQL Server, 16-18, 445-446
  databases
    connecting to, 236-256
    creating applications on, 239-249
  intrinsic database, 446-447
  supported editions, 237
SQL Server Reporting Services, 597-599
SSME (SQL Server Management Studio Express), 256-273
  SQL Azure, connecting via, 256-273
StackOverflow, 768
Start Page, 25-26
starter kits, 766
static spans, logic tier, 454-455
steps, debugging, 522-524
stored procedures
  .NET methods, mapping, 727-736
  WCF RIA Services, calling through, 726-745
String data type, 48
string resources, controls, adding, 658-659
string-length validation, data-centric applications, 86-87
Structured Query Language (SQL), 171
styling
  applications, 348-349
  command panel, 641-644
  Navigation area, 650-653
  root container, 641
  Screen Content area, 644-650
  User Information area, 653
summary properties, 73-76
  testing, 76

tabbed windows, arranging, 37-38
Table Designer, 45-46
Table Layout control, 138
tables, 45
  binding to screens, 253-256
  versus entities, 46
Tabs Layout control, 138
target web servers, configuring, 374-376
tasks, optimization, 1
Team Foundation Server (TFS). See TFS (Team Foundation Server)
team projects
  automating builds, 499-506
  creating, 483-484
  source control, submitting to, 485-492
  version control, 499
technologies, applications, 444-445
Telerik, 608
templates
  screen
    creating, 670-691
    details, 687-690
    testing, 685-686
  screens
    adding local screen members, 690-691
    IScreenTemplate properties, 673-675
test projects, creating, 509-510
testing
  applications, credentials, 327-329, 334-335
  custom controls, 667-669
  custom data sources, 746-749
  custom shells, 654-655
  custom themes, 633-634
  extensions, 703-705
  screen templates, 685-686
  summary properties, 76

How can we make this index more useful? Email us at indexes@samspublishing.com
Text and Picture control, 138
TextBox control, 54
TFS (Team Foundation Server)
  applications, managing life cycles, 480-506
  connecting to, 481-482
  team projects
    automating builds, 499-506
    creating, 483-484
    submitting to source control, 483-484
    version control, 499
  work items, creating and assigning, 494-497
Thalman, Matt, 766
Theme property (Application Designer), 345
themes, 229, 619-620
  creating, 621-634
  custom, testing, 633-634
  editing, 628-629
    Expression Blend 4, 630-631
    extensibility projects, adding to, 625-628
    green, creating, 631-633
    Metro, 621
theming service, presentation tier, 465
third-party extensions, 769-770
threads, analyzing, Threads window, 534
Threads window, analyzing threads, 534
three-tier applications, 341-344
  deploying, 373-392
  publishing, 377-379
    MSDeploy packages, 385-392
    to web servers, 379-383
three-tier desktop clients, running applications as, 83-84
tiers, applications
  data access, 445-452
  data source, 445-452
  logic, 444-462
  presentation, 445, 462-469
time, displaying, 89
Tips & Tricks community, 768
toolbars, customizing, 544-545
toolkits, 21
topologies, deployment, selecting, 353
trace points, 527-532
training kits, 766
two-tier applications, 341-344
  deploying, 352-373
    deployment topologies, 353
    Publish Application Wizard, 352-368
  publishing, 368-373
types
  business, creating, 693-705
  controls, 657
U
unit testing, helper code, 506-512
updates
  entities, Bing Maps, 576
  extensions, releasing, 757-758
uploading VSIX package, Visual Studio Gallery, 753-756
user controls
  reports, creating, 609-612
  screens, binding to, 569-572
User Information area, styling, 653
user interfaces
  business-oriented, 115-146
  controls, 53-55
    designing, 663-664
  data entry screens, creating, 116
  permissions, logic, 307-312
  reports, displaying, 612-613
screens, implementing, 52-60
search screens, creating, 116-118

user roles
creating, 312-329
permissions, 320-323

user settings
exporting, 547-548
importing, 548-550
managing, 546-550

users, roles, assigning, 323-327

V
validation
built-in rules, 152-153
client, 154-167
custom rules, writing, 153-174
data, 175
customizing, 149
data types, 160-165
data validation, implementing, 146-148, 696-700
data-centric applications
default validation of business types, 91-93
input-data validation, 85-90
entity collections, 169-170
entity properties, 156-167
master-details relationships, 147-148, 170-172
model, 150-153
multiple validation issues, 85
rule types, 151
rules, .NET Framework, 172-174
single entity, 167-169
validation framework, logic tier, 456
validation rules, 456-457
ValidationSeverity enumeration, 159
Validator Factory class, 699
variable values, displaying, Locals window, 530
VB (Visual Basic) 6, 6-8
view models, 640-641
viewer controls, creating, 701-703
Visual Basic 6, 6-8
Visual Basic 2010 Unleashed, 22
Visual FoxPro, 9-10
Visual Studio 2010
custom controls, creating, 563-572
LightSwitch, 39-40
Visual Studio Express, 11-13
Visual Studio Gallery, 767
VSIX package, uploading to, 753-756
Visual Studio LightSwitch Help website, 767-768
Visual Studio .NET, 10-11
visualizers, debuggers, 535-536
VSIX package
signing, 752
uploading to Visual Studio Gallery, 753-756

W
Watch windows, 533-534
WCF (Windows Communications Foundation), 709
WCF RIA Services
calling from applications, 720-726
creating, 727
custom data sources, creating and using, 705-745
deploying, 719
.NET Framework, 709-711
stored procedures, calling through, 726-745
XML data, 711-719

How can we make this index more useful? Email us at indexes@samspublishing.com
web browsers, applications, running in, 84
web servers
  configuring, 374-376
  three-tier applications, publishing to, 379-383
WebAddress.lsml Definition File listing (19.1), 695
Wilson, Glenn, 768
Window menu, 38
windows
  arranging, 37-38
  managing, 37-38
Windows authentication, 338
  implementing, 101, 294-300
Windows Azure
  configuring
    role instances, 407
    services, 398-402
    SQL, 402-404
  connecting to, 394-398
  deploying applications to, 392-407
Windows Communication Foundation (WCF), 709
Word documents, exporting data to, 586-592
work items, creating and assigning, TFS (Team Foundation Server), 494-497
writing custom rules, validation, 153-174

XAML (Extensible Application Markup Language), 620
XML data, WCF RIA Services, 711-719
XtraReports, 600-605