Expert C# 2008
Business Objects

Rockford Lhotka
To my Mom and Dad. Thank you for all you’ve taught me through the years!
Contents at a Glance

About the Author ........................................................................................................... xix
About the Technical Reviewers .................................................................................... xxi
Acknowledgments .......................................................................................................... xxiii
Introduction ...................................................................................................................... xxv

CHAPTER 1  Distributed Architecture ................................................................. 1
CHAPTER 2  Framework Design .......................................................................... 37
CHAPTER 3  Object-Oriented Application Design .................................................. 101
CHAPTER 4  CSLA .NET Object Stereotypes ....................................................... 143
CHAPTER 5  CSLA .NET Object Templates .......................................................... 173
CHAPTER 6  Business Framework Implementation ................................................ 207
CHAPTER 7  Property Declarations ................................................................. 241
CHAPTER 8  Object Status Management .............................................................. 263
CHAPTER 9  Parent-Child Relationships ............................................................... 273
CHAPTER 10 Data Binding .................................................................................. 287
CHAPTER 11 Business and Validation Rules .......................................................... 331
CHAPTER 12 Authentication and Authorization .................................................... 353
CHAPTER 13 N-Level Undo .............................................................................. 369
CHAPTER 14 LINQ to CSLA .............................................................................. 399
CHAPTER 15 Persistence and the Data Portal ....................................................... 419
CHAPTER 16 Other Framework Features ............................................................... 487
CHAPTER 17 Business Object Implementation .................................................... 517
CHAPTER 18 Example Data Access .................................................................. 547
CHAPTER 19 Windows Presentation Foundation UI .......................................... 583
CHAPTER 20 Web Forms UI .............................................................................. 619
CHAPTER 21 WCF Service Interface .................................................................. 675

INDEX ................................................................................................................................. 719
Contents

About the Author ................................................................. xix
About the Technical Reviewers ............................................ xxi
Acknowledgments ............................................................... xxiii
Introduction ........................................................................... xxv

CHAPTER 1  Distributed Architecture ........................................ 1

Logical and Physical Architecture ........................................ 1
N-Tier and SOA ................................................................. 3
Complexity ....................................................................... 4
Relationship Between Logical and Physical Models ............... 4
A 5-Layer Logical Architecture ........................................... 9
Applying the Logical Architecture ....................................... 14
The Way Ahead ................................................................. 19
Managing Business Logic .................................................... 19
Potential Business Logic Locations ...................................... 20
Business Objects ............................................................... 23
Mobile Objects ................................................................. 26
Architectures and Frameworks ............................................. 34
Conclusion ........................................................................... 35

CHAPTER 2  Framework Design .................................................. 37

Basic Design Goals ............................................................ 38
Validation and Business Rules ............................................ 39
Tracking Whether the Object Has Changed .......................... 41
Integrated Authorization .................................................... 41
Strongly Typed Collections of Child Objects ......................... 42
N-Level Undo Capability ..................................................... 42
Simple and Abstract Model for the UI Developer ................... 45
Supporting Data Binding ..................................................... 50
Object Persistence and Object-Relational Mapping ............... 54
Custom Authentication ........................................................ 60
Designing the Framework ................................................. 61
  Business Object Creation ............................................. 61
  N-Level Undo Functionality ......................................... 69
  Data Binding Support .................................................. 72
  Business and Validation Rules ....................................... 74
  Data Portal ................................................................... 77
  Custom Authentication .................................................. 91
  Integrated Authorization ................................................ 92
  Helper Types and Classes ............................................... 93
Namespace Organization .................................................... 96
Conclusion ................................................................... 98

CHAPTER 3  Object-Oriented Application Design .................. 101

  Responsibility-Driven Design ......................................... 102
    Use-Case or Story-Based Analysis .................................. 102
    Objects with Responsibilities ....................................... 102
    Objects Exist for Use Cases ....................................... 103
    Normalization of Behavior ......................................... 103
  Application Requirements ............................................. 104
    Use Cases .................................................................. 105
  Object Design ............................................................ 108
    Initial Design ................................................................ 108
    Revising the Design ................................................... 110
    Custom Authentication .............................................. 122
  Using CSLA .NET .......................................................... 123
  Database Design .......................................................... 126
    Creating the Databases .............................................. 127
    PTracker Database ..................................................... 129
    Security Database ...................................................... 139
Conclusion ................................................................... 140
CHAPTER 4  CSLA .NET Object Stereotypes ........................................... 143
  Basic Terminology and Object Graph Structure .................................. 144
  Business Object Life Cycle .............................................................. 147
    Object Creation ........................................................................... 147
    Object Retrieval ......................................................................... 152
    Updating Editable Objects .................................................................. 156
    Disposing and Finalizing Objects ..................................................... 162
  Business Class Structure .................................................................... 163
    The Serializable or DataContract Attribute ..................................... 164
    Common Regions ........................................................................... 166
    Non-public Default Constructor ....................................................... 170
  Conclusion ...................................................................................... 171

CHAPTER 5  CSLA .NET Object Templates .............................................. 173
  Business Class Structure .................................................................... 174
    Editable Root Business Objects ....................................................... 174
    Editable Child Business Objects ...................................................... 180
    Switchable Objects .......................................................................... 184
    Editable Root Collection ................................................................ 187
    Editable Child Collection ................................................................ 190
    Read-Only Business Objects ............................................................ 191
    Read-Only Child Objects .................................................................. 193
    Read-Only Collection ....................................................................... 194
    Read-Only Child Collection .............................................................. 196
    Command Objects ........................................................................... 197
    Name/Value List Objects .................................................................. 198
    Dynamic Editable Collection ............................................................. 199
    Dynamic Editable Root Objects ........................................................ 201
    Criteria Objects .............................................................................. 202
  Conclusion ...................................................................................... 205
# CHAPTER 6  Business Framework Implementation  

CSLA .NET Project Structure  
- Project Directory Structure  
- Project Settings  
- Project Signing  
- Supporting Localization  

Csla Namespace  
- ApplicationContext  
- BusinessBase  
- BusinessListBase  
- CommandBase  
- CriteriaBase  
- DataPortal  
- EditableRootListBase  
- NameValueListBase  
- PropertyInfo  
- ReadOnlyBase  
- ReadOnlyListBase  
- SingleCriteria  
- SmartDate  
- Utilities  

Csla.Core Namespace  
- BusinessBase  
- ExtendedBindingList  
- IBusinessObject Interface  
- ICommandObject Interface  
- IEditableBusinessObject Interface  
- IEditableCollection Interface  
- IReadOnlyObject Interface  
- IReadOnlyCollection Interface  
- ISavable Interface  
- ISmartField Interface  
- ISupportUndo Interface  
- ITrackStatus Interface  
- IUndoableObject Interface  
- ObjectCloner Class  
- ReadOnlyBindingList  

Conclusion
WPF ................................................................. 312
   Object Data Binding ........................................ 312
   Collection Data Binding ................................... 314
   Controls and Helper Objects .............................. 315
Web Forms ...................................................... 327
   Controls and Helper Objects .............................. 327
Conclusion .................................................... 329

CHAPTER 11 Business and Validation Rules ......................... 331

   Types of Rules .............................................. 331
   Csla.Validation Namespace ............................... 332
      RuleHandler Delegate .................................. 332
      RuleArgs Class .......................................... 333
      DecoratedRuleArgs Class .............................. 334
      RuleMethod Class ...................................... 334
      RuleDescription Class ................................ 336
      ValidationRules Class .................................. 337
      BrokenRule Class ....................................... 343
      BrokenRulesCollection Class ......................... 344
      ValidationException ................................... 346
   Common Validation Rules ................................. 347
      CommonRules ............................................ 347
Conclusion .................................................... 351

CHAPTER 12 Authentication and Authorization ....................... 353

   Authentication ............................................. 353
      Windows Authentication ................................ 355
      Custom Authentication ................................. 355
   Authorization .............................................. 360
      Type Level Authorization ............................... 361
      Property and Method Level Authorization ........... 364
Conclusion .................................................... 368
### CHAPTER 13  N-Level Undo

- Using Undo .................................................. 370
- Implementing Undo .......................................... 370
  - ISupportUndo Interface .................................. 372
  - NotUndoableAttribute Class ............................. 372
  - UndoableBase Class ....................................... 373
  - BusinessBase Class ....................................... 382
  - BusinessListBase Class .................................. 385
- Conclusion ................................................... 397

### CHAPTER 14  LINQ to CSLA

- Reducing Code with LINQ .................................. 399
- Overview of LINQ to CSLA .NET ............................ 400
  - Binding to Results from LINQ to Objects ............... 400
  - Indexed LINQ Queries .................................... 400
- LINQ and Projection ........................................ 401
  - Identity Projections and LinqBindingList<T> .......... 401
  - Understanding LinqBindingList ......................... 401
- Overview of Indexed Search Using CSLA .NET ........... 403
  - Serialization and Indexing ............................... 403
  - Index Mode ................................................ 403
- The IQueryable Implementation for CSLA .NET .......... 404
  - Understanding Expression Trees ....................... 405
  - Digging into IQueryProvider ............................ 406
  - LinqBindingList ........................................... 408
- Indexed LINQ and CSLA .NET ............................... 408
  - Managing the Index Set .................................. 409
  - Expression Evaluation .................................... 410
  - The Indexing Object Model ............................... 411
- Conclusion ................................................... 417
CHAPTER 15  Persistence and the Data Portal ........................................ 419

Data Portal Design ............................................................... 420
   Separation of Business Logic and Data Access ................ 420
   Consistent Coding Model for Root and Child Objects .......... 421
   Channel Adapter and Message Router Patterns ................. 421
   Distributed Transaction Support ................................. 427
   Context and Location Transparency ......................... 428
   Authorizing Server Calls ........................................... 431
   Asynchronous Behaviors ......................................... 432
   Object Factories .................................................... 433

Base Class Support ......................................................... 434
   Factory Methods and Criteria ............................... 436
   Save Methods ..................................................... 437
   Updating Child Objects with the Field Manager ............. 441
   Updating Editable Collections ............................ 441

Reflection and Dynamic Method Invocation ....................... 442
   The MethodCaller Class .................................... 443
   The LateBoundObject Class .................................. 445

Channel Adapter ............................................................ 445
   The RunLocal Attribute ...................................... 445
   The DataPortal Class ....................................... 446
   The DataPortal<T> Class ................................. 455
   The IDataPortalServer Interface ......................... 457
   The IDataPortalProxy Interface ....................... 458
   The LocalProxy Class ....................................... 458
   The WcfProxy Class ........................................ 459
   The WcfPortal Class ......................................... 460

Distributed Transaction Support ........................................ 462
   The Transactional Attribute ................................ 462
   The CsIA.Server.DataPortal Object .................... 462
   The ServicedDataPortal Class ......................... 466
   The TransactionalDataPortal Class .............. 467

Message Router .......................................................... 468
   The DataPortalSelector Class ......................... 469
   The SimpleDataPortal Class ......................... 470
   The FactoryDataPortal Class ....................... 476
   The FactoryLoader Property ......................... 476
   The ChildDataPortal Class ......................... 479
Context and Location Transparency ........................................... 480
  The DataPortalContext Class ........................................... 480
  The DataPortalResult Class ........................................... 484
  Cs1a.Server.DataPortalException ....................................... 484
Conclusion ........................................................................ 485

CHAPTER 16  Other Framework Features ................................. 487

Date Handling with SmartDate ............................................. 488
  Initializing the Struct ....................................................... 490
  Supporting Empty Dates .................................................... 491
  Conversion Functions ....................................................... 491
  Text Functions ................................................................ 492
  Date Functions ................................................................ 493
  Database Format ............................................................. 494
Data Access ...................................................................... 494
  Managing Database Connections and Contexts .................... 495
  SafeDataReader ................................................................ 499
  DataMapper .................................................................... 503
Windows Workflow Foundation ............................................ 508
  Starting a Workflow from an Object .................................... 510
  WorkflowManager Class .................................................. 510
Conclusion ........................................................................ 516

CHAPTER 17  Business Object Implementation ...................... 517

ProjectTracker Objects ......................................................... 517
Setting Up the Project .......................................................... 518
Business Class Implementation ............................................. 519
  Project ........................................................................... 520
  ProjectResources ............................................................. 530
  ProjectResource .............................................................. 532
  Assignment .................................................................... 534
  RoleList ......................................................................... 536
Resource and Related Objects ............................................. 536
  ProjectList and ResourceList ............................................ 538
Roles ............................................................................. 539
  Role ............................................................................. 541
Implementing Exists Methods .............................................. 543
Contents

CHAPTER 18  Example Data Access ................................. 547

Data Access Layer Design ........................................ 547
  Data Access Models ............................................. 548
  Balancing Design Issues ..................................... 554
Data Access Objects ............................................. 556
  Using LINQ to SQL ............................................. 556
  The ProjectTracker.Dal.Linq Project ....................... 557
Business Class Implementation ............................... 559
  Project .......................................................... 560
  ProjectResources ............................................. 569
  ProjectResource .............................................. 571
  RoleList ......................................................... 574
  ProjectList and ResourceList ............................... 575
  Roles ........................................................... 577
  Implementing Exists Methods ............................... 579
Conclusion ......................................................... 581

CHAPTER 19  Windows Presentation Foundation UI ........................... 583

Custom Authentication in WPF ................................ 584
Interface Design .................................................. 585
  User Control Framework .................................... 587
  Value Converters .............................................. 591
  Application Configuration ................................... 593
  PTWpf Project Setup ......................................... 596
  The MainForm Window ....................................... 596
  The Login Window ............................................ 600
  The RolesEdit Form .......................................... 602
  The ResourceList Form ...................................... 609
  The ProjectList Form ........................................ 611
  The ProjectEdit Form ........................................ 612
Conclusion ......................................................... 617
CHAPTER 20  Web Forms UI ......................................................... 619
  Web Development and Objects ............................................. 619
  State Management .............................................................. 622
    State on the Web Server ................................................ 622
    Transferring State to or from the Client .............................. 625
    State in a File or Database ............................................ 626
  Interface Design ............................................................... 627
    Application Configuration ............................................... 629
    PTWeb Site Setup .......................................................... 633
    Master Page ................................................................. 634
    Login Page .................................................................. 638
  Business Functionality ....................................................... 646
    RolesEdit Form ............................................................... 646
    ProjectList Form ............................................................ 656
    ProjectEdit Form ............................................................ 661
  Conclusion .................................................................. 673

CHAPTER 21  WCF Service Interface ............................................. 675
  Choosing Between Client/Server and SOA .............................. 675
  Overview of WCF Services .................................................. 677
    Elements of a WCF Service .............................................. 679
    Custom Authentication ................................................. 685
  Designing a WCF Service Interface .................................... 701
    Service Design .............................................................. 702
    Application Configuration ............................................. 703
    PTWcfService Site Setup .............................................. 705
    Service Contract and Implementation .............................. 706
  Web Service Consumer Implementation .............................. 711
    Generating a WCF Information Page ................................. 711
    A Simple Smart Client ................................................... 713
  Conclusion .................................................................. 718

INDEX .................................................................................. 719
ROCKFORD LHOTKA is the author of numerous books. He is a Microsoft Regional Director, a Microsoft Most Valuable Professional (MVP), and an INETA speaker. He contributes to several major magazines and presents regularly at major conferences around the world, including Microsoft Tech Ed and VS Live. Rockford is the Principal Technology Evangelist for Magenic (www.magenic.com), one of the nation’s premier Microsoft Gold Certified Partners that is focused on delivering business value through applied technology. For more information, go to www.lhotka.net.
About the Technical Reviewers

**JAMES MILLER** is a senior architect and technical evangelist specializing in enterprise solutions. He has worked in multiple industries and capacities in both the public and private sectors, and he has more than 25 years of programming experience under his belt. He has embraced CSLA since 2002 and has been an active proponent of the framework in his career and on the CSLA forums. He is currently working with an ISV, leading an international group of developers to upgrade their products to the latest .NET technologies, practices, tools, and techniques, while espousing the virtues of OOP, SOA, TDD, agile development, and CSLA. Jim is a proud graduate of the University of Michigan, is Microsoft certified in both VB .NET and C# for Windows and Web-based applications, and jumped at the chance to contribute to the next version of the CSLA framework.

Jim lives in a rural area outside Ann Arbor, Michigan, with his wife, five children, four cats, and three dogs. He fills much of his limited free time as the head coach of his local high school’s junior varsity boys’ lacrosse team. Jim still has an electric guitar plugged in over in a corner, a bookshelf filled with tech books, and a comfortable chair on the deck, perfect for viewing the deer as they meander by.

**ANDRÉS VILLANUEVA** is a consultant/developer living in Argentina. After a start in IT at age 15, he moved to the software industry, coding with Visual FoxPro and Visual Basic 6. In 2004, Andrés moved on to .NET and hasn’t looked back. His early software experiences were in the banking industry, where he quickly rose as a leader, helping his firm improve consistency by implementing the CSLA framework. Since those days, he has made the leap into the consulting world and now provides software services from his office in Argentina to various clients around the world. He is an open source software fan and the current lead on the CslaGenerator project—an open source code-generation tool that targets development on the CSLA framework. In his little free time, Andrés enjoys playing soccer and relaxing with jazz music.

**JOE FALLON** is the Director of Framework Development at PurchasingNet, Inc., and is responsible for the development and implementation of the .NET Framework for PNet products. Prior to joining PurchasingNet, he worked at Nestle Chocolate and Confection as an assistant plant engineer after completing five years of service in the U.S. Army as a captain in the field artillery. During his tenure at Nestle USA, he held various positions in industrial engineering and IT.

Joe graduated from the United States Military Academy at West Point, New York, and was the 1981 recipient of the General Omar Bradley Award as the Academy’s no. 1 mathematics major. He has been a Microsoft MVP for eight years in a row.
This book is a major update to the previous edition. This book, and CSLA .NET 3.6, exist thanks to a lot of work from many people.

I need to acknowledge the support, patience, and love from my wife and sons over the past many years. Without you, this would have been impossible.

I’d also like to thank Greg Frankenfield and Paul Fridman for making Magenic such an awesome place to work. The support that you and the rest of Magenic have provided has been great, and I appreciate it very much. It is an honor to work with everyone there.

CSLA .NET 3.6 is the result of a lot of work by several Magenic colleagues, including Sergey Barskiy, Justin Chase (now at Microsoft), Aaron Erickson, and Nermin Dibek. Jon Stonecash, Mark Steinberg, Grant Breems, and Chris Williams contributed as well. Sandy Fougerousse created the CSLA .NET for Silverlight logo.

A number of people outside Magenic also contributed to CSLA .NET 3.6, including Ricky Supit, Mark Chesney, and Miguel Castro. The CSLA .NET for Windows logo was contributed by Chris Russi.

The Apress editorial team went above and beyond to help shape this book into what you see here and to help get it done as rapidly as possible. I owe them all a debt of gratitude for their fine work.

Finally, I’d like to thank the scores of people who’ve sent me emails, posted on the forum and my blog with messages of support and encouragement, or just plain asked when the book would be done. The great community that has grown around these books and the CSLA .NET framework is wonderful, and I thank you all. I hope you find this book to be as rewarding to read as it has been for me to write.

Code well and have fun!
Introduction

I have a passion for frameworks. In more than 20 years as a professional developer, I’ve never worked on a computing platform that did everything I needed it to do to build applications productively. The Microsoft .NET platform is wonderful, but it doesn’t always do quite what I want or need. To address those needs, I’m always looking for tools and frameworks, and sometimes I end up creating them myself.

A framework is simply the codification of an architecture or design pattern. Before you can have a good framework, you need to have an architecture. That means you need to have a vision and a set of goals both for the architecture and the kinds of applications it should enable.

This book is about application architecture, design, and development in .NET using object-oriented concepts. The focus is on creating business objects and implementing them to work in various distributed environments, including web and client/server configurations. The book makes use of a great many .NET technologies, object-oriented design and programming concepts, and distributed architectures.

Much of the book walks through the thought process I used in designing and creating the CSLA .NET framework to support object-oriented application development in .NET. This includes a lot of architectural concepts and ideas. It also involves some in-depth use of advanced .NET techniques to create the framework.

The book also shows how to make use of the framework to build a sample application with several different interfaces. If you wish, you could skip the framework design chapters and simply make use of the framework to build object-oriented applications.

One of my primary goals in creating the CSLA .NET framework was to simplify .NET development. Developers using the framework in this book don’t need to worry about the details of underlying technologies such as remoting, serialization, or reflection. All of these are embedded in the framework, so that a developer using it can focus almost entirely on business logic and application design rather than on getting caught up in “plumbing” issues.

This book is a major update to the previous edition, Expert C# 2005 Business Objects. This updated book takes advantage of new features of .NET 3.5 and applies lessons learned by using .NET 2.0 and 3.0 over the past few years.

This book is the most recent expression of concepts I’ve been working on for more than a dozen years. My goal all along has been to enable the productive use of object-oriented design in distributed n-tier applications. Over the years, both the technologies and my understanding and expression of the concepts have evolved greatly.

From CSLA .NET 2.0 to 3.6

Over the past eight years, the CSLA .NET framework has become one of the most widely used development frameworks on the Microsoft .NET platform. Since I introduced the .NET version in 2001, the framework has grown and evolved quite a lot, in part due to changes to the .NET platform itself, and in part due to feedback from the vibrant community surrounding CSLA .NET.

The CSLA .NET framework is a reflection of an underlying architecture I call CSLA, for component-based, scalable, logical architecture. Over the years, I’ve received hundreds of emails from people who have used CSLA as a basis for their own architectures, as they’ve built applications ranging from small, single-user programs to full-blown enterprise applications that power major parts of their businesses.
This framework addresses two primary areas of object-oriented software development:

- How to use business objects to efficiently build Windows, web, and service-oriented applications
- How to enable the use of object-oriented design in a distributed computing environment

While .NET supports the use of objects, the author of an object has to do a lot of work to fully support important .NET concepts such as data binding. Much of the focus of CSLA .NET and of this book is on enabling objects to fully support data binding, as well as on other important concepts such as validation and authorization. For most users of CSLA .NET, these are the primary benefits that the framework provides.

Many people build distributed n-tier or service-oriented applications. Using object-oriented design and business objects in a distributed environment has its own challenges, and CSLA .NET uses various techniques to overcome those challenges. For n-tier client/server applications, the framework supports the idea of mobile objects—objects that actually move between computers in an n-tier environment. Mobile objects provide a powerful way to implement object-oriented designs in distributed environments. For service-oriented applications, CSLA .NET can be used to build both edge applications and services. The framework is compelling for edge application creation and is often useful for creating services or workflow activities as well.

As the .NET platform and the CSLA .NET framework have evolved, I’ve made a great many changes and added many new features. In some cases, using the new concepts and features has required making changes to existing business objects and user interface code. I don’t take backward compatibility lightly, yet it is important to advance the concepts to keep up with both changes in technology and my views on both object-oriented and distributed computing.

When possible, I have minimized the impact on existing code, so the transition shouldn’t be overly complex for most applications. Although there are a few breaking changes from version 3.0 to 3.6, most existing code should upgrade easily. Even version 2.1 code should upgrade with relative ease. Business classes written with CSLA .NET versions 1.x or 2.0 will require quite a bit of effort to bring forward.

Over the years, I’ve received a handful of emails from people for whom CSLA .NET wasn’t successful, but this isn’t surprising. To use CSLA .NET effectively, you must become versed in object-oriented design, understand the concept of mobile objects, and develop a host of other skills. The mobile object architecture has many benefits, but it’s not the simplest or the easiest to understand.

However, over that same period of time, I’ve received countless emails from people who have had tremendous success in building applications using CSLA .NET. These applications range from Windows to web, from small to enterprise, from retail to manufacturing to military environments. I am amazed, pleased, and humbled by these emails and by all the cool places where CSLA .NET has helped organizations and individuals around the world.

**Designing CSLA .NET**

One of the characteristics of .NET is that it often provides several ways to solve the same problem. Some of the available approaches are better than others, but the best one for a given problem may not be immediately obvious. Over the past eight years, I’ve spent a lot of time researching many of these options and techniques. Although a variety have proven to work, in the end I’ve arrived at the one that best matches my original goals.

I have a specific set of goals for the architecture and the book. These goals are important, because they’re key to understanding why I made many of the choices I did in terms of which .NET technologies to use and how to use them. The goals are as follows:
• To support a fully object-oriented programming model
• To allow the developer to use the architecture without jumping through hoops
• To enable high scalability
• To enable high performance
• To enable developer productivity when using business objects, including:
  • Support for data binding in Windows and Web Forms
  • Support for many types of UIs based on the same objects
  • Management of validation rules
  • Management of authorization rules
  • N-level undo on a per-object basis (edit, cancel, apply)
  • Integration with distributed transaction technologies such as Enterprise Services and System.Transactions
• To support the use of object-oriented design in a distributed environment through the use of mobile objects
• To simplify .NET by handling complex issues such as serialization, reflection, and network communication
• To use the tools provided by Microsoft—notably IntelliSense and the Autocomplete feature in Visual Studio .NET

Of these, saving the developer from jumping through hoops—that is, allowing him or her to do “normal” programming—has probably had the largest impact. To meet all these goals without a framework, the developer would have to write a lot of extra code to track business rules, implement n-level undo, and support serialization of object data. All this code is important, but it adds nothing to the business value of the application.

Fortunately, .NET offers some powerful technologies that help to reduce or eliminate much of this “plumbing” code. If those technologies are then wrapped in a framework, a business developer shouldn’t have to deal with them at all. In several cases, this goal of simplicity drove my architectural decisions. The end result is that the developer can, for the most part, simply write a standardized C# class and have it automatically enjoy all the benefits of n-level undo, business rule tracking, and so forth.

It has taken a great deal of time and effort, but I’ve certainly enjoyed putting this architecture and this book together, and I hope that you will find both valuable during the development of your own applications.

Framework License

LICENSE AND WARRANTY
The CSLA .NET framework is Copyright 2008 by Rockford Lhotka.
You can use this Software for any noncommercial purpose, including distributing derivative works. You can use this Software for any commercial purpose other than you may not use it, in whole or in part, to create a commercial framework product.
In short, you can use CSLA .NET and modify it to create other commercial or business software, you just can’t take the framework itself, modify it and sell it as a product.
In return, the owner simply requires that you agree:
This Software License Agreement (“Agreement”) is effective upon your use of CSLA .NET (“Software”).
1. Ownership.
The CSLA .NET framework is Copyright 2008 by Rockford Lhotka, Eden Prairie, MN, USA.

2. Copyright Notice.
You must not remove any copyright notices from the Software source code.

3. License.
The owner hereby grants a perpetual, non-exclusive, limited license to use the Software as set forth in this Agreement.

If you distribute the Software in source code form you must do so only under this License (i.e. you must include a complete copy of this License with your distribution).

5. Binary or Object Distribution.
You may distribute the Software in binary or object form with no requirement to display copyright notices to the end user. The binary or object form must retain the copyright notices included in the Software source code.

You may not sell the Software. If you create a software development framework based on the Software as a derivative work, you may not sell that derivative work. This does not restrict the use of the Software for creation of other types of non-commercial or commercial applications or derivative works.

7. Disclaimer of Warranty.
The Software comes “as is”, with no warranties. None whatsoever. This means no express, implied, statutory or other warranty, including without limitation, warranties of merchantability or fitness for a particular purpose, noninfringement, or the presence or absence of errors, whether or not discoverable. Also, you must pass this disclaimer on whenever you distribute the Software.

8. Liability.
Neither Rockford Lhotka nor any contributor to the Software will be liable for any of those types of damages known as indirect, special, consequential, incidental, punitive or exemplary related to the Software or this License, to the maximum extent the law permits, no matter what legal theory it's based on. Also, you must pass this limitation of liability on whenever you distribute the Software.

If you sue anyone over patents that you think may apply to the Software for a person's use of the Software, your license to the Software ends automatically. The patent rights, if any, licensed hereunder only apply to the Software, not to any derivative works you make.

10. Termination.
Your rights under this License end automatically if you breach it in any way. Rockford Lhotka reserves the right to release the Software under different license terms or to stop distributing the Software at any time. Such an election will not serve to withdraw this Agreement, and this Agreement will continue in full force and effect unless terminated as stated above.

This Agreement shall be construed and enforced in accordance with the laws of the state of Minnesota, USA.

12. No Assignment.
Neither this Agreement nor any interest in this Agreement may be assigned by Licensee without the prior express written approval of Developer.
13. **Final Agreement.**
   This Agreement terminates and supersedes all prior understandings or agreements on the subject matter hereof. This Agreement may be modified only by a further writing that is duly executed by both parties.

14. **Severability.**
   If any term of this Agreement is held by a court of competent jurisdiction to be invalid or unenforceable, then this Agreement, including all of the remaining terms, will remain in full force and effect as if such invalid or unenforceable term had never been included.

15. **Headings.**
   Headings used in this Agreement are provided for convenience only and shall not be used to construe meaning or intent.

**What You Need to Use This Book**

The code in this book has been verified to work against Microsoft Visual Studio 2008 Professional Edition SP1 and against version 3.5 SP1 of the .NET Framework. The database is a SQL Server Express database, which is included with Visual Studio 2008 Professional. The Enterprise version of Visual Studio 2008 and the full version of SQL Server are useful but not necessary.

In order to run the tools and products listed previously, you'll need at least one PC with Windows Vista, Windows XP SP2 (or higher), Windows Server 2003, or Windows Server 2008 installed. To test CSLA .NET’s support for multiple physical tiers, of course, you’ll need an additional PC (or you can use Virtual PC or a similar tool) for each tier that you wish to add.

**How This Book Is Structured**

This book covers the thought process behind the CSLA .NET for Windows version 3.6 architecture. It describes the construction of the framework that supports the architecture, and it demonstrates how to create WPF, Web Forms, and WCF service applications based on business objects written using the framework.

If you are reading this book to understand the process of designing and constructing a development framework for the .NET platform, then you should read all chapters. If you are reading this book to understand how to use the CSLA .NET framework and are less interested in how the framework itself is designed and implemented, then you should read Chapters 1 through 5 and Chapters 17 through 21.

Chapter 1 introduces some of the concepts surrounding distributed architectures, including logical and physical architectures, business objects, and distributed objects. Perhaps more importantly, this chapter sets the stage, showing the thought process that results in the remainder of the book.

Chapter 2 takes the architecture described at the end of Chapter 1 and uses it as the starting point for a code framework that enables the goals described earlier. By the end of the chapter, you’ll have seen the design process for the objects that will be implemented in Chapters 6 through 16; but before that, there’s some other business to attend to.

In Chapter 3, I discuss the basics of responsibility-driven object-oriented design. As an example, this chapter lays out the requirements and design for a sample application.

Chapters 4 and 5 discuss how to use each of the primary base classes in the CSLA .NET framework to create your own business objects. I discuss in detail the object-oriented stereotypes supported by the CSLA .NET base classes, along with the code structure for editable and read-only objects, and collections and name/value lists.

Chapters 6 through 16 are all about the construction of the CSLA .NET framework itself. If you’re interested in the code behind property declarations, validation rules, authorization rules, n-level undo, mobile object support, and object persistence, then these are the chapters for you. In addition,
they make use of some of the more advanced and interesting parts of the .NET Framework, including data binding, serialization, reflection, dynamic method invocation, WCF, .NET security, Enterprise Services, System.Transactions, strongly named assemblies, dynamically loaded assemblies, application configuration files, and more.

Chapters 17 and 18 create the business objects for the application. These chapters illustrate how you can use the framework to create a powerful set of business objects rapidly and easily for an application. The end result is a set of objects that not only model business responsibilities, but also support data binding, validation, authorization, n-level undo, and various physical configurations that can optimize performance, scalability, security, and fault tolerance, as discussed in Chapter 1.

Chapter 19 demonstrates how to create a WPF interface to the business objects. Chapter 20 covers the creation of an ASP.NET Web Forms interface with comparable functionality.

Chapter 21 shows how to build WCF services using business objects. This approach enables service-oriented development by providing a programmatic interface to the business objects that any web service or WCF client can call.

By the end, you’ll have a framework that supports object-oriented application design in a practical, pragmatic manner. The framework implements a logical model that you can deploy in various physical configurations to optimally support Windows, web, and XML service clients.

**Downloading the Code**

The code that reflects the contents of this book is available in the Source Code/Download area of the Apress website (www.apress.com). For the latest version of the framework and the example application, visit www.lhotka.net/cslanet/download.aspx.

**Contacting the Author**

You may reach Rockford Lhotka on his website, www.lhotka.net, which contains his blog, information about the framework and book, and his contact information.
CHAPTER 1

Distributed Architecture

Object-oriented design and programming are big topics—entire books are devoted solely to the process of object-oriented design or to using object-oriented programming in various languages and on various programming platforms. My focus in this book isn’t to teach the basics of object-oriented design or programming, but rather to show how you may apply them to the creation of distributed .NET applications.

It can be difficult to apply object-oriented design and programming effectively in a physically distributed environment. This chapter is intended to provide a good understanding of the key issues surrounding distributed computing as it relates to object-oriented development. I’ll cover a number of topics, including the following:

- How logical n-layer architectures help address reuse and maintainability
- How physical n-tier architectures impact performance, scalability, security, and fault tolerance
- The difference between data-centric and object-oriented application models
- How object-oriented models help increase code reuse and application maintainability
- The effective use of objects in a distributed environment, including the concepts of anchored and mobile objects
- The relationship between an architecture and a framework

This chapter provides an introduction to the concepts and issues surrounding distributed object-oriented architecture. Then, throughout this book, I’ll be exploring an n-layer architecture that may be physically distributed across multiple machines. I’ll show how to use object-oriented design and programming techniques to implement a framework supporting this architecture. I’ll create a sample application that demonstrates how the architecture and the framework support development efforts.

Logical and Physical Architecture

In today’s world, an object-oriented application must be designed to work in a variety of physical configurations. Even the term application has become increasingly blurry due to all the hype around service-oriented architecture (SOA). If you aren’t careful, you can end up building applications by combining several applications, which is obviously confusing.

When I use the term application in this book, I’m referring to a set of code, objects, or components that’s considered to be part of a single, logical unit. Even if parts of the application are in different .NET assemblies or installed on different machines, all the code will be viewed as being part of a singular application.
This definition works well when describing most traditional application models, such as single-tier or 2-tier rich client applications, n-tier smart client applications, web applications, and so forth. In all those cases, the application consists of a set of objects or components that are designed to work together within the context of the application.

You can contrast this with an SOA model, where multiple services (each essentially a separate application) interact through message-based communication. In an SOA model, the idea is to build an enterprise system that is composed of applications and services. In this context, both applications and services are stand-alone, autonomous units of functionality, which means they both meet the definition of an application. Confusingly enough, this means a service is merely an application that has an XML interface instead of an HTML or graphical interface.

If you’re thinking about service-oriented systems as you read this book, the term application means one of two things. First, it may refer to a service implementation. Second, it may refer to an application on the edge of the system that allows users to interact with the system. Edge applications are much like traditional applications, except they typically interact with services instead of databases for retrieving and storing data.

You can contrast the traditional and SOA models with a workflow model, which you’re likely to encounter when using Windows Workflow Foundation (WF). In this environment, an application is often implemented (in whole or part) in the form of a workflow. However, the workflow itself merely orchestrates a set of activities, and each activity should be an autonomous, stand-alone unit of functionality. This means that an activity must meet the definition of an application. An activity is merely an application that has no real user interface beyond the data-binding infrastructure built into WF.

Traditional, service-oriented and workflow applications might run on a single machine. However, it’s very likely that they will run on multiple machines, such as a web server or a smart client and an application server. Given these varied physical environments, you’re faced with the following questions:

- Where do the objects reside?
- Are the objects designed to maintain state, or should they be stateless?
- How is object-relational mapping handled when retrieving or storing data in the database?
- How are database transactions managed?

Before discussing some answers to these questions, it’s important that you fully understand the difference between a physical architecture and a logical architecture. After defining these terms, I’ll define objects and mobile objects, and show you how they fit into the architectural discussion.

When most people talk about n-tier applications, they’re talking about physical models in which the application is spread across multiple machines with different functions: a client, a web server, an application server, a database server, and so on. And this isn’t a misconception—these are indeed n-tier systems. The problem is that many people tend to assume there’s a one-to-one relationship between the layers (tiers) in a logical model and the tiers in a physical model, when in fact that’s not always true.

A physical n-tier architecture is quite different from a logical n-layer architecture. An n-layer architecture has nothing to do with the number of machines or network hops involved in running the application. Rather, a logical architecture is all about separating different types of functionality. The most common logical separation is into an Interface layer, a Business layer, and a Data layer. These may exist on a single machine or on three separate machines—the logical architecture doesn’t define those details.

**Note** There is a relationship between an application’s logical and physical architectures: the logical architecture always has at least as many layers as the physical architecture has tiers. There may be more logical layers than physical tiers (because one physical tier can contain several logical layers), but never fewer.
The sad reality is that many applications have no clearly defined logical architecture. Often the logical architecture merely defaults to the number of physical tiers. This lack of a formal, logical design causes problems because it reduces flexibility. If a system is designed to operate in two or three physical tiers, then changing the number of physical tiers at a later date is typically very difficult. However, if you start by creating a logical architecture of three layers, you can switch more easily between one, two, or three physical tiers later on.

Additionally, having clean separation between these layers makes your application more maintainable, because changing one layer often has minimal impact on the other layers. Nowhere is this more true than with the Interface layer (sometimes called the UI or Presentation layer), where the ability to switch between Windows Presentation Foundation (WPF), Windows Forms, Web Forms, ASP.NET MVC, and workflow and service-based interfaces is critical.

The flexibility to choose your physical architecture is important because the benefits gained by employing a physical n-tier architecture are different from those gained by employing a logical n-layer architecture. A properly designed logical n-layer architecture provides the following benefits:

- Logically organized code
- Easier maintenance
- Better reuse of code
- Better team-development experience
- Higher clarity in coding

On the other hand, a properly chosen physical n-tier architecture can provide the following benefits:

- Performance
- Scalability
- Fault tolerance
- Security

It goes almost without saying that if the physical or logical architecture of an application is designed poorly, there will be a risk of damaging the things that would have been improved had the job been done well.

N-Tier and SOA

It is important to realize that a physical service-oriented architecture is not the same as an n-tier architecture. In fact, the two concepts can be complementary. It is also important to know that the concept of a logical n-layer architecture is the same in SOA as in any other type of application model.

In logical n-layer models, a service should have the same layers as any other application: Interface, Business, and Data. In a logical n-layer model, the Interface layer consists of XML messages, but that’s not a lot different from the HTML used in a web-based Interface layer. The Business layer is much the same as in any other application; it contains the business logic and behaviors that make the service useful. The data layer is also much the same as in any other application, in that it stores and retrieves data as necessary.

However, the physical n-tier model might not appear to translate to the SOA world at all. Some people would say that SOA makes n-tier concepts totally obsolete, but I disagree. SOA has an important set of goals around loose coupling, reuse of functionality, and open communication. An n-tier client/server architecture has a complementary set of goals around performance, avoiding duplication of code, and targeted functionality. The reality is that both models are useful, and they complement each other.
For example, you might use a service-oriented model to create a service that is available on the Internet. However, the service implementation might be n-tier, with the service interface on the web server and parts of the business implementation running on a separate application server. The result is a reusable service that enjoys high performance and security and avoids duplication of code.

**Complexity**

Experienced designers and developers often view a good n-tier architecture as a way of simplifying an application and reducing complexity, but this isn’t necessarily the case. It’s important to recognize that n-tier designs are typically more complex than single-tier designs. Even novice developers can visualize the design of a form or a page that retrieves data from a file and displays it to the user, but novice developers often struggle with 2-tier designs and are hopelessly lost in an n-tier environment.

With sufficient experience, architects and developers do typically find that the organization and structure of an n-tier model reduces complexity for large applications. However, even a veteran n-tier developer will often find it easier to avoid n-tier models when creating a simple form to display some simple data.

The point here is that n-tier architectures only simplify the process for large applications or complex environments. They can easily complicate matters if all you’re trying to do is create a small application with a few forms that will be running on someone’s desktop computer. (Of course, if that desktop computer is one of hundreds or thousands in a global organization, then the environment may be so complex that an n-tier solution provides simplicity.)

In short, n-tier architectures help to decrease or manage complexity when any of these are true:

- The application is large or complex.
- The application is one of many similar or related applications that, when combined, may be large or complex.
- The environment (including deployment, support, and other factors) is large or complex.

On the other hand, n-tier architectures can increase complexity when all of these are true:

- The application is small or relatively simple.
- The application isn’t part of a larger group of enterprise applications that are similar or related.
- The environment isn’t complex.

Something to remember is that even a small application is likely to grow, and even a simple environment often becomes more complex over time. The more successful your application, the more likely that one or both of these will happen. If you find yourself on the edge of choosing an n-tier solution, it’s typically best to go with it. You should expect and plan for growth.

This discussion illustrates why n-tier applications are viewed as relatively complex. A lot of factors—both technical and nontechnical—must be taken into account. Unfortunately, it isn’t possible to say definitively when n-tier does and doesn’t fit. In the end, it’s a judgment call that you, as an application architect, must make, based on the factors that affect your particular organization, environment, and development team.

**Relationship Between Logical and Physical Models**

Some architectures attempt to merge logical n-layer and physical n-tier concepts. Such mergers seem attractive because they seem simpler and more straightforward, but typically they aren’t good in practice—they can lead people to design applications using a logical or physical architecture that isn’t best suited to their needs.
The Logical Model

When you’re creating an application, it’s important to start with a logical architecture that clarifies the roles of all components, separates functionality so that a team can work together effectively, and simplifies overall maintenance of the system. The logical architecture must also include enough layers so that you have flexibility in choosing a physical architecture later on.

Traditionally, you would devise at least a 3-layer logical model that separates the interface, the business logic, and the data-management portions of the application. Today that’s rarely sufficient, because the “interface” layer is often physically split into two parts (browser and web server), and the “logic” layer is often physically split between a client or web server and an application server. Additionally, various application models have been used to break the traditional Business layer into multiple parts—model-view-controller (MVC) and facade-data-logic being two of the most popular at the moment.

This means that the logical layers are governed by the following rules:

- The logical architecture includes layers in order to organize components into discrete roles.
- The logical architecture must have at least as many layers as the anticipated physical deployment will have tiers.

Following these rules, most modern applications have four to six logical layers. As you’ll see, the architecture used in this book includes five logical layers.

Cross-Layer Communication

Just because an application is organized into layers doesn’t mean those layers can be deployed arbitrarily on different tiers. The code in one layer communicates with the layer immediately above or below it in the architecture. If you don’t design that communication properly, it may be impossible to put a network (tier) boundary between the layers.

For example, the boundary between the Business layer and the Data layer is often highly optimized. Most applications have a network boundary between the Data layer and the rest of the application, so modern data access technologies are good at optimizing cross-network communication in this scenario.

The boundary between the Interface layer and the Business layer is often not optimized for this purpose. Many applications make use of data binding, which is a “chatty” technology involving many property, method, and event calls between these two layers. The result is that it is often impractical and undesirable to put a network boundary between these layers.

Not all layer boundaries should be designed to enable a tier boundary. You should design an architecture up front to enable the potential for tier boundaries in certain locations and to disallow them in other cases. If done properly, the result is a balance between flexibility and capability.

The Physical Model

By ensuring that the logical model has enough layers to provide flexibility, you can configure your application into an appropriate physical architecture that will depend on your performance, scalability, fault tolerance, and security requirements. The more physical tiers included, the worse the performance will be; however, there is the potential to increase scalability, security, and/or fault tolerance.

Performance and Scalability

The more physical tiers there are, the worse the performance? That doesn’t sound right, but if you think it through, it makes perfect sense: performance is the speed at which an application responds to a user. This is different from scalability, which is a measure of how performance changes as load (such as increased users) is added to an application. To get optimal performance—that is, the fastest
possible response time for a given user—the ideal solution is to put the client, the logic, and the data on the user’s machine. This means no network hops, no network latency, and no contention with other users.

If you decide that you need to support multiple users, you might consider putting application data on a central file server. (This is typical with Access and dBASE systems, for example.) However, this immediately affects performance because of contention on the data file. Furthermore, data access now takes place across the network, which means you’ve introduced network latency and network contention, too. To overcome this problem, you could put the data into a managed environment such as SQL Server or Oracle. This will help to reduce data contention, but you’re still stuck with the network latency and contention problems. Although improved, performance for a given user is still nowhere near what it was when everything ran directly on that user’s computer.

Even with a central database server, scalability is limited. Clients are still in contention for the resources of the server, with each client opening and closing connections, doing queries and updates, and constantly demanding the CPU, memory, and disk resources that other clients are using. You can reduce this load by shifting some of the work to another server. An application server, possibly running Enterprise Services or Internet Information Services (IIS), can provide database connection pooling to minimize the number of database connections that are opened and closed. It can also perform some data processing, filtering, and even caching to offload some work from the database server.

Note It is important to realize that modern database servers can often easily handle hundreds of concurrent users in a 2-tier architecture. For most applications, scalability is not a good reason to move from a 2- to 3-tier model.

These additional steps provide a dramatic boost to scalability, but again at the cost of performance. The user’s request now has two network hops, potentially resulting in double the network latency and contention. For a single user, the system gets slower; however, it is able to handle many times more users with acceptable performance levels.

In the end, the application is constrained by the most limiting resource. This is typically the speed of transferring data across the network—but if the database or application server is underpowered, it can become so slow that data transfer across the network won’t be an issue. Likewise, if the application does extremely intense calculations and the client machines are slow, then the cost of transferring the data across the network to a relatively idle high-speed server can make sense.

Security

Security is a broad and complex topic, but by narrowing the discussion solely to consider how it’s affected by physical n-tier decisions, it becomes more approachable. The discussion is no longer about authentication or authorization as much as it is about controlling physical access to the machines on which portions of the application will run. The number of physical tiers in an application has no impact on whether users can be authenticated or authorized, but physical tiers can be used to increase or decrease physical access to the machines on which the application executes.

For instance, in a 2-tier Windows Forms or ASP.NET application, the machine running the interface code must have credentials to access the database server. Switching to a 3-tier model in which the data access code runs on an application server means that the machine running the interface code no longer needs those credentials, potentially making the system more secure.

Security requirements vary radically based on the environment and the requirements of your application. A Windows Forms application deployed only to internal users may need relatively little security, but an ASP.NET application exposed to anyone on the Internet may need extensive security.

To a large degree, security is all about surface area: how many points of attack are exposed from the application? The surface area can be defined in terms of domains of trust.