

Professional

# ADO.NET 3.5

## with LINQ and the Entity Framework

Roger Jennings



*Updates, source code, and Wrox technical support at [www.wrox.com](http://www.wrox.com)*



Programmer to Programmer™

# Get more out of **WROX.com**

## Interact

Take an active role online by participating in our P2P forums

## Wrox Online Library

Hundreds of our books are available online through Books24x7.com

## Wrox Blox

Download short informational pieces and code to keep you up to date and out of trouble!

## Chapters on Demand

Purchase individual book chapters in pdf format

## Join the Community

Sign up for our free monthly newsletter at [newsletter.wrox.com](http://newsletter.wrox.com)

## Browse

Ready for more Wrox? We have books and e-books available on .NET, SQL Server, Java, XML, Visual Basic, C#/ C++, and much more!

## Contact Us.

We always like to get feedback from our readers. Have a book idea? Need community support? Let us know by e-mailing [wrox-partnerwithus@wrox.com](mailto:wrox-partnerwithus@wrox.com)

# **Professional**

## **ADO.NET 3.5 with LINQ and the Entity Framework**

Introduction .....	xxvii
<b>Part I: Getting a Grip on ADO.NET 3.5</b>	
Chapter 1: Taking a New Approach to Data Access in ADO.NET 3.5.....	3
<b>Part II: Introducing Language Integrated Query</b>	
Chapter 2: Understanding LINQ Architecture and Implementation.....	63
Chapter 3: Executing LINQ Query Expressions with LINQ to Objects .....	91
Chapter 4: Working with Advanced Query Operators and Expressions.....	155
<b>Part III: Applying Domain-Specific LINQ Implementations</b>	
Chapter 5: Using LINQ to SQL and the LinqDataSource.....	195
Chapter 6: Querying DataTables with LINQ to DataSet .....	243
Chapter 7: Manipulating Documents with LINQ to XML .....	267
Chapter 8: Exploring Third-Party and Emerging LINQ Implementations .....	317
<b>Part IV: Introducing the ADO.NET Entity Framework</b>	
Chapter 9: Raising the Level of Data Abstraction with the Entity Data Model.....	357
Chapter 10: Defining Storage, Conceptual, and Mapping Layers .....	393
Chapter 11: Introducing Entity SQL .....	433
<b>Part V: Implementing the ADO.NET Entity Framework</b>	
Chapter 12: Taking Advantage of Object Services and LINQ to Entities.....	469
Chapter 13: Updating Entities and Complex Types .....	503
Chapter 14: Binding Entities to Data-Aware Controls .....	533
Chapter 15: Using the Entity Framework as a Data Source.....	567
Index .....	607



**Professional**

**ADO.NET 3.5 with LINQ and the Entity  
Framework**



**Professional**

**ADO.NET 3.5 with LINQ and the Entity  
Framework**

Roger Jennings



**Wiley Publishing, Inc.**

# Professional ADO.NET 3.5 with LINQ and the Entity Framework

Published by

Wiley Publishing, Inc.  
10475 Crosspoint Boulevard  
Indianapolis, IN 46256  
[www.wiley.com](http://www.wiley.com)

Copyright © 2009 by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada

ISBN: 978-0-470-18261-1

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Cataloging-in-Publication Data

Jennings, Roger.

Professional ADO.NET 3.5 with LINQ and the Entity Framework / Roger Jennings.

p. cm.

Includes index.

ISBN 978-0-470-18261-1 (paper/website)

1. ActiveX. 2. Microsoft LINQ. 3. Database design. 4. Microsoft .NET. 5. Query languages (Computer science) I. Title.

QA76.9.D26J475 2009

006.7'882—dc22

2008048201

No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, except as permitted under Sections 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 646-8600. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at <http://www.wiley.com/go/permissions>.

**Limit of Liability/Disclaimer of Warranty:** The publisher and the author make no representations or warranties with respect to the accuracy or completeness of the contents of this work and specifically disclaim all warranties, including without limitation warranties of fitness for a particular purpose. No warranty may be created or extended by sales or promotional materials. The advice and strategies contained herein may not be suitable for every situation. This work is sold with the understanding that the publisher is not engaged in rendering legal, accounting, or other professional services. If professional assistance is required, the services of a competent professional person should be sought. Neither the publisher nor the author shall be liable for damages arising herefrom. The fact that an organization or Web site is referred to in this work as a citation and/or a potential source of further information does not mean that the author or the publisher endorses the information the organization or Web site may provide or recommendations it may make. Further, readers should be aware that Internet Web sites listed in this work may have changed or disappeared between when this work was written and when it is read.

For general information on our other products and services please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

**Trademarks:** Wiley, the Wiley logo, Wrox, the Wrox logo, Wrox Programmer to Programmer, and related trade dress are trademarks or registered trademarks of John Wiley & Sons, Inc. and/or its affiliates, in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. Wiley Publishing, Inc. is not associated with any product or vendor mentioned in this book.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

*This book is dedicated to my wife, Alexandra.*



## About the Author

**Roger Jennings** is an author and consultant specializing in Microsoft .NET *n*-tier and client/server database applications and data-intensive Windows Communication Foundation (WCF) Web services. He's been a beta tester for all versions of Visual Basic and Visual Studio, starting with the Professional Extensions for Visual Basic 2.0 (code named Rawhide).

More than 1.25 million copies of Roger's 25 computer-oriented books are in print, and they have been translated into more than 20 languages. He's the author of *Expert One-on-One Visual Basic 2005 Database Programming* for Wiley/Wrox, three editions of *Database Developer's Guide to Visual Basic* (SAMS Publishing), two editions of *Access Developer's Guide* (SAMS), 11 editions of *Special Edition Using Microsoft Access* (QUE Publishing), and two editions of *Special Edition Using Windows NT 4.0 Server* (QUE). He's also written developer-oriented books about Windows 3.1 multimedia, Windows 95, Windows 2000 Server, Active Directory Group Policy, Visual Basic Web services, and Microsoft Office InfoPath 2003 SP-1. Roger has been a contributing editor of Redmond Media Group's *Visual Studio Magazine* and its predecessor, *Visual Basic Programmer's Journal* for almost 15 years. His articles also appear in *Redmond Magazine* and he writes "TechBriefs" and cover stories for *Redmond Developer News*.

Roger has more than 30 years of computer-related experience, beginning with real-time medial data acquisition and chemical process control systems driven by Wang 700 calculators and later Wang BASIC microcomputers. He is the principal developer for OakLeaf Systems, a Northern California software consulting firm and author of the OakLeaf Systems blog (<http://oakleafblog.blogspot.com>). His OakLeaf Code of Federal Regulations (CFR) ASP.NET Web service demonstration won the charter Microsoft .NET Best Award for Horizontal Solutions (<http://www.microsoft.com/presspass/features/2002/aug02/08-07netwinners.mspx>).



# Credits

**Executive Editor**

Robert Elliott

**Development Editor**

Adaobi Obi Tulton

**Technical Editor**

Oren Novotny

**Senior Production Editor**

Debra Banninger

**Copy Editor**

Foxxe Editorial Services

**Editorial Manager**

Mary Beth Wakefield

**Production Manager**

Tim Tate

**Vice President and Executive Group Publisher**

Richard Swadley

**Vice President and Executive Publisher**

Joseph B. Wikert

**Project Coordinator, Cover**

Lynsey Stanford

**Proofreader**

Amy Morales, Word One New York

**Indexer**

Jack Lewis



# Acknowledgments

Many thanks to Oren Novotny, this book's technical editor, for corrections and suggestions as the chapters passed through Community Technical Previews and beta releases of Visual Studio 2008 SP1. Oren is senior C# / .NET developer at Lava Trading, a financial services organization in New York City. He's also the developer of LINQ to Streams (a.k.a., SLinq or Streaming LINQ) for processing continuous data streams, such as stock tickers or sensor data. The project's home page on CodePlex (<http://www.codeplex.com/SLinq>) provides downloadable source code and includes an animated GIF simulation of a stock ticker displayed in a DataGridView.

Thanks to Microsoft Principal Architect Matt Warren for his highly detailed and very helpful blog posts about LINQ to SQL and its development history, as well as a virtuoso demonstration of implementing the `IQueryable<T>` interface (<http://blogs.msdn.com/mattwar>). Program manager Dinesh Kulkarni provided beta testers with numerous LINQ to SQL insights and tips. Daniel Simmons treated early Entity Framework (EF) adopters to advice, technical insight, code samples, and FAQs for the Data Programmability team's flagship object/relational modeling (O/RM) tool. Pablo Castro, originally lead developer on the EF team, gave vision to "Project Astoria" and later became the ADO.NET Data Services Framework's architect. I'm especially indebted to Pablo for conceiving the "Transparent Design" initiative for Astoria, which Program Manager Tim Mallalieu wisely adopted for EF v2. Finally, thanks to all the members of the LINQ, LINQ to SQL, and EF teams who patiently answered my and other users' questions in the LINQ Project General, and ADO.NET Entity Framework and LINQ to Entities (Prerelease) forums.

Joe Wikert, Wiley Technical Publishing's vice president and publisher, and Executive Editor Bob Elliott convinced me to start writing .NET developer books for Wiley/Wrox. Adaobi Obi Tulton, development editor for this book and my earlier *Expert One-on-One Visual Basic Database Programming* title, made sure that chapters didn't slip too far behind the release-to-manufacturing (RTM) date of Visual Studio 2008 SP1, which included the final implementation of Entity Framework and Entity Data Model v1. Debra Banninger, senior production editor, fixed many grammatical and stylistic lapses. I appreciate their contributions, as well as those of all others in the production process, to the book's completion.



# Contents

## Introduction

xxvii

## Part I: Getting a Grip on ADO.NET 3.5

<b>Chapter 1: Taking a New Approach to Data Access in ADO.NET 3.5</b>	<b>3</b>
<b>Language Integrated Query (LINQ)</b>	<b>5</b>
LINQ to Objects	8
Sample LINQ Queries	9
LINQ Data Binding with the ToList() Operator	12
LINQ to SQL	15
Mapping Tables to Entities with the LINQ to SQL OR Designer	15
Examining the Generated Classes	16
Binding Data-Enabled Controls to Entity Data Sources	21
Programming the DataContext	23
Emulating Joins by Navigating Associations	24
Loading Child Objects Lazily or Eagerly	27
Creating Explicit Joins with LINQ Join . . . On Expressions	27
Applying Operators to Emulate SQL Functions, Modifiers, and Operators	30
Updating Table Data	30
Generating a SQL Server Database with the OR Designer	34
LinqDataSource Control	34
LINQ to DataSets	34
Joining DataSets	35
Navigating Entity Set Associations	37
LINQ to Entities	38
LINQ to XML	39
LINQ to XML's Objects	39
Querying XML Documents	40
Transforming or Creating XML Documents	44
LINQ to XSD	47
<b>The ADO.NET Entity Framework and Entity Data Model</b>	<b>48</b>
Mapping from the Physical to the Conceptual Schema	50
Creating the Default EDM with the Entity Data Model Wizard	52
Modifying Storage to Conceptual Mapping with the EDM Designer	53

## Contents

---

Creating and Binding to a Data Source from a Data Model	54
Materializing an Object Context	56
<b>    Summary</b>	<b>57</b>
 <b>Part II: Introducing Language Integrated Query</b>	
<b>Functional Languages, Haskell, and Cω</b>	<b>60</b>
<b>A Brief History of LINQ</b>	<b>61</b>
<b>Chapter 2: Understanding LINQ Architecture and Implementation</b>	<b>63</b>
<b>Namespaces That Support LINQ in .NET Fx 3.5</b>	<b>64</b>
<b>C# and VB Extensions to Support LINQ</b>	<b>66</b>
Implicitly Typed Local Variables	67
Object Initializers	68
Array Initializers with Object Initializers	70
Collection Initializers	70
Anonymous Types	71
Extension Methods	72
Anonymous Methods and Generic Predicates	75
Lambda Expressions	77
C# 3.0 Lambda Expressions	78
VB 9.0 Lambda Expressions	79
Standard Query Operators	80
C# 3.0 Query Expressions	80
Lazy and Eager Evaluation	82
VB 9.0 Query Expressions	83
Expression Trees and Compiled Queries	83
C# 3.0 Expression Trees	83
VS 2008 SP1's ExpressionTreeVisualizer	84
VB 9.0 Expression Trees	86
Compiled Queries	87
The IQueryable<T> Interface and Domain-Specific LINQ Implementations	88
<b>        Summary</b>	<b>89</b>
<b>Chapter 3: Executing LINQ Query Expressions with LINQ to Objects</b>	<b>91</b>
<b>Standard Query Operators by Group</b>	<b>92</b>
<b>SQOs as Keywords in C# 3.0 and VB 9.0</b>	<b>93</b>
<b>The LINQ Project Sample Query Explorers</b>	<b>95</b>
<b>Sample Classes for LINQ to Objects Code Examples</b>	<b>96</b>
C# Class Definition and Initialization Code Example	98

---

VB Class Definition and Initialization Code Example	100
<b>Restriction Operator: Where</b>	<b>101</b>
Simple Where Expressions	102
Compound Where Expressions	102
Method Calls with Index Arguments and Use of IndexOf()	104
<b>Projection Operators</b>	<b>105</b>
Select	106
Simple Select Projection Expression	106
Multiple Select Projection Expression with Index Value	107
Index Value Expressions with the Let Keyword	107
SelectMany	108
Basic SelectMany Implementation for Associated Objects	108
SelectMany Overload for Equi-Joins of Associated Objects	110
<b>Partitioning Operators</b>	<b>111</b>
Take	112
Skip	112
Skip/Take Example	112
TakeWhile	113
SkipWhile	114
SkipWhile/TakeWhile Example	114
<b>Join Operators</b>	<b>115</b>
Join	116
GroupJoin	117
<b>Concatenation Operator: Concat</b>	<b>120</b>
<b>Ordering Operators</b>	<b>121</b>
OrderByDescending	122
ThenBy	122
ThenByDescending	123
Reverse	123
Ordering Operator Examples	123
<b>Grouping Operator: GroupBy</b>	<b>125</b>
GroupBy with Method Call Syntax	126
GroupBy with Query Expression Syntax	127
<b>Set Operators</b>	<b>128</b>
Distinct	129
Union	129
Intersect	131
Except	132
<b>Conversion operators</b>	<b>132</b>
AsEnumerable	133
AsQueryable	133
Cast	135

# Contents

---

OfType	136
To . . . Operators	138
ToArray	138
ToList	139
To Dictionary	139
ToLookup	141
<b>Equality Operator: SequenceEqual</b>	<b>142</b>
<b>Element operators</b>	<b>144</b>
First, FirstOrDefault	144
Last, LastOrDefault	145
Single, SingleOrDefault	145
DefaultIfEmpty	146
ElementAt, ElementAtOrDefault	146
<b>Generation Operators</b>	<b>147</b>
Range	147
Repeat	147
Empty	147
<b>Quantifier Operators</b>	<b>148</b>
Any	148
All	148
Contains	149
<b>Aggregate Operators</b>	<b>149</b>
Count and LongCount	150
Min, Max, Sum, and Average	151
Aggregate	152
<b>Summary</b>	<b>153</b>
<b>Chapter 4: Working with Advanced Query Operators and Expressions</b>	<b>155</b>
<b>Exploring Basic Query Syntax for Aggregate Operators</b>	<b>156</b>
Basic Method Call Syntax with Numerical Operators	157
Expression Syntax with Let as a Temporary Local Aggregate Variable	158
Using a Lambda Function for Aggregation of Child Entity Values	158
Using Visual Basic's Aggregate . . . Into Keywords	159
Formatting the Query Output	159
<b>Using Group By with Aggregate Queries</b>	<b>160</b>
Grouping with Associated Child Objects	161
Using the Aggregate Keyword and Into Expression with VB	161
Using C# Group By Expression Syntax	162
Grouping with Joined Child Objects	163
Combining Join and Group By Operations with Hierarchical Group Join Expressions	164
Comparing Group Joins with Nested LINQ Queries	166
Emulating Left Outer Joins with Entity Associations	168

---

<b>Taking Full Advantage of the Contains() SQO</b>	<b>169</b>
Emulating SQL Where Clauses with Compound OR Operators	169
Emulating the SQL IN() Function with Contains()	171
<b>Compiling Query Expression Trees to Improve Performance</b>	<b>172</b>
<b>Mocking Collections for Testing LINQ to SQL and LINQ to Entities Projects</b>	<b>176</b>
Creating Mock Object Classes and Initializers	177
Creating Object Graphs with GroupJoin Expressions	182
<b>Summary</b>	<b>187</b>

## Part III: Applying Domain-Specific LINQ Implementations

<b>LINQ to SQL</b>	<b>191</b>
<b>LINQ to DataSet</b>	<b>193</b>
<b>LINQ to XML</b>	<b>193</b>

### Chapter 5: Using LINQ to SQL and the LinqDataSource **195**

<b>Object/Relational Mapping with LINQ to SQL</b>	<b>197</b>
Mapping Tables to Entity Sets with the LINQ to SQL O/R Designer	197
Generating Partial Entity Classes and Mapping Files with SqlMetal.exe	200
Working with *.dbml and *.xml Mapping Files	203
Editing *.dbml Files in the Designer	203
Editing *.xml Mapping Files in an XML Editor	205
Examining the Generated Classes	206
Instantiating the DataContext and Its Object Graph	210
<b>Using LINQ to SQL as a Data Access Layer</b>	<b>212</b>
The LINQ to SQL Query Pipeline	212
Adding, Updating, and Removing Objects	215
Detecting and Resolving Concurrency Conflicts	219
Substituting Stored Procedures for Dynamic SQL	222
Using a SELECT Stored Procedure to Return an ISingleResult< TEntity >	223
Using INSERT, UPDATE, and DELETE Stored Procedures	224
Moving the LINQ to SQL Layer to a Middle Tier	225
<b>ASP.NET Databinding with the LinqDataSource Control</b>	<b>226</b>
Adding a LinqDataSource to a Page	226
Substituting EntityRef for ForeignKey Values in Databound Web Controls	228
Eager-Loading EntityRef Values to Reduce Database Server Traffic	231
<b>Databinding Windows Form Controls to Entities</b>	<b>233</b>
Autogenerating the Obligatory Hierarchical Data Editing Form	233
Persisting Entity Edits and Collection Modifications	236
Adding Members to a Collection with a Surrogate, Autoincrementing Primary Key	237
Deleting Members with a Dependent EntitySet from a Collection	238
<b>Summary</b>	<b>242</b>

# Contents

---

<b>Chapter 6: Querying DataTables with LINQ to DataSet</b>	<b>243</b>
<b>Comparing DataSets and DataContexts</b>	<b>244</b>
<b>Exploring LINQ to DataSet Features</b>	<b>246</b>
<b>Running Read-Only LINQ to DataSet Queries</b>	<b>247</b>
Querying Untyped DataSets	248
Customizing Lookup Lists	253
Querying Typed DataSets	254
<b>Creating LinqDataViews for DataBinding with AsDataView()</b>	<b>257</b>
<b>Copying LINQ Query Results to DataTables</b>	<b>260</b>
Copying Typed DataRows	262
Processing Anonymous Types from Projections	263
<b>Summary</b>	<b>266</b>
 <b>Chapter 7: Manipulating Documents with LINQ to XML</b>	 <b>267</b>
<b>Integrating XML into the CLR</b>	<b>267</b>
Minimizing XML/Object Mismatch with Xen	268
Querying XML with C#	269
<b>The System.Xml.Linq Namespace</b>	<b>269</b>
<b>Querying Basic XML Infosets</b>	<b>271</b>
Inferring a Schema and Enabling IntelliSense for VB Queries	273
Taking Advantage of VB 9.0 Axis Properties	275
Implicit versus Explicit Typing of Element and Attribute Content	276
<b>Composing XML Infosets</b>	<b>278</b>
Using Functional Construction with C# 3.0	280
Using Literal XML Construction with VB 9.0	284
<b>Grouping Elements and Aggregating Numeric Values of Business Documents</b>	<b>288</b>
Using GroupJoin to Produce Hierarchical Documents	289
Taking Advantage of 1:Many and Many:1 Associations	290
Aggregating Order_Details and Orders Subtotals per Customer	292
<b>Working with XML Namespaces and Local Names</b>	<b>295</b>
XML Namespaces in C# 3.0	297
Namespaces in C# LINQ to XML Queries	297
Removing Expanded Namespaces from C# Queries	298
Functionally Constructing C# XDocuments with Multiple Namespaces	300
XML Namespaces in VB 9.0	302
Enabling IntelliSense for Namespaces	302
Multiple Namespaces in XML Literal Queries	304
Transforming Documents That Have Multiple Namespaces, with Literal XML Code	306
<b>Performing Heterogeneous Joins and Lookup Operations</b>	<b>308</b>
Using Lookup Operations to Add Child Element Groups	308

Joining Documents to Insert Elements	312
Joining Documents and LINQ to SQL or LINQ to Object Entities	314
<b>Summary</b>	<b>316</b>
<b>Chapter 8: Exploring Third-Party and Emerging LINQ Implementations</b>	<b>317</b>
<b>Emerging Microsoft LINQ Implementations</b>	<b>318</b>
Parallel LINQ	318
Programming with PLINQ	319
Processing Queries	321
Running the PLINQ Samples	322
LINQ to REST	325
Adopting URIs as a Query Language	328
Running the Sample Northwind.svc WCF Service and Client Projects	329
Processing Services Requests with the NwindServicesClient	336
LINQ to XSD	340
LINQ to XSD's History	341
LINQ to Stored XML	341
<b>Third-Party Domain-Specific LINQ Implementations</b>	<b>342</b>
LINQ to Active Directory	342
LINQ to SharePoint	344
<b>Summary</b>	<b>349</b>
<b>Part IV: Introducing the ADO.NET Entity Framework</b>	
<b>A Brief History of Entity Framework's Development</b>	<b>353</b>
<b>Entity Framework's Future</b>	<b>354</b>
<b>The ADO.NET Entity Framework Vote of No Confidence</b>	<b>355</b>
<b>Chapter 9: Raising the Level of Data Abstraction with the Entity Data Model</b>	<b>357</b>
<b>Understanding the Entity-Relationship Model</b>	<b>359</b>
Entity-Relationship and EDM Terminology	360
Entity-Relationship Diagrams	361
<b>Comprehending Entity Framework Architecture and Components</b>	<b>363</b>
Mapping from the Physical to the Conceptual Layer with the EDM Designer	365
Creating the XML Mapping Files and Object Layer Class File with the EDM Wizard	365
Adding, Updating, and Deleting the Model's Objects	367
Editing EntityType and AssociationSet Properties	368
Analyzing the <b>ModelName.edmx</b> File's Sections	370
Scanning the StorageModels Group	370

# Contents

---

Examining the ConceptualModels Group	371
Tracing the Mappings Group	373
<b>Working with the Entity Client, Entity SQL and Client Views</b>	<b>375</b>
Writing EntityQueries in Entity SQL	376
Executing Entity SQL Queries as Client Views	376
<b>Taking Advantage of Object Services</b>	<b>381</b>
Working with ObjectContext	382
Creating an Object Context and ObjectQuery	382
MetadataWorkspace	383
ObjectStateManager	384
Writing ObjectQueries with Entity SQL	384
Composing ObjectQueries with Query Builder Methods	387
Using the LINQ to Entities Provider	388
<b>Understanding the Persistence Ignorance Controversy</b>	<b>390</b>
<b>Summary</b>	<b>391</b>
<b>Chapter 10: Defining Storage, Conceptual, and Mapping Layers</b>	<b>393</b>
<b>Exploring and Customizing the EDMX File</b>	<b>395</b>
Storage Models (SSDL Content)	397
The EntityContainer Subgroup	398
EntityType Subgroups	400
The Product EntityType Subgroup	402
Function Subelements and Subgroups	402
Association Subgroups	409
Conceptual Models (CSDL Content)	411
The EntityContainer Subgroup	411
The EntityType Subgroup	413
The Association Subgroup	414
Mapping (MSL Content)	414
The EntitySetMapping Subgroup	415
The AssociationSetMapping Subgroup	417
The FunctionImportMapping Element	418
<b>Implementing Table-per-Hierarchy Inheritance</b>	<b>419</b>
Specifying the Discriminator Column and Creating a Derived Class	419
Querying Base and Derived Classes	421
Incorrect Results from is, Is, TypeOf, and OfType() Operators	423
Type Discrimination in Entity SQL Queries	424
Disambiguate Derived Object Types with an Abstract Base Type	425
<b>Traversing the MetadataWorkspace</b>	<b>427</b>
<b>Summary</b>	<b>431</b>

---

<b>Chapter 11: Introducing Entity SQL</b>	<b>433</b>
<b>Using the eSqlBlast Entity SQL Query Utility</b>	<b>434</b>
Connection Page	434
Model Page	435
Query Page	436
Result Page	437
<b>Understanding How Entity SQL Differs from Transact-SQL</b>	<b>439</b>
Entity Alias Prefixes Are Mandatory	440
Explicit Projections Are Required	441
The VALUE Modifier Flattens Results	441
Dot-Notation Syntax Returns Many:One Navigation Properties	442
Nested Queries Are Required for One:Many Navigation Properties	444
JOINS Are the Last Resort	445
NAVIGATE Is a Complex Substitute for Dot Notation or Nested Queries	445
REF, DEREF, CREATEREF, ROW, and KEY Manage Entity References	446
Type Constructors Create ROWs, Multisets, and Instances of EntityTypes	449
The UNION, INTERSECT, OVERLAPS, and EXCEPT Set Operators Require Sub-Queries	450
Sorting Collections Returned by Set Operators Requires a Nested Query	451
Set Operators ANYELEMENT and FLATTEN Work on Collections	452
SKIP and LIMIT Sub-Clauses of the ORDER BY Clause Handle Paging	452
IS OF, OFTYPE, and TREAT Are Type Operators for Polymorphic Queries	453
Subqueries That Return Aggregate Values for WHERE Clause Constraints	
Throw Exceptions	454
<b>Executing eSQL Queries against the EntityClient</b>	<b>455</b>
Parsing the IExtendedDataRecord from an EntityDataReader	456
Measuring the Performance Penalty of EntitySQL Queries	459
Executing Parameterized eSQL Queries	460
<b>Using SQL Server Compact as an Entity Framework Data Store</b>	<b>461</b>
Substituting SSCE for SQL Server [Express] as a Data Store	462
<b>Summary</b>	<b>463</b>

## Part V: Implementing the ADO.NET Entity Framework

<b>Entity Framework vs. LINQ to SQL</b>	<b>466</b>
<b>Entity Framework Futures</b>	<b>467</b>
<b>Chapter 12: Taking Advantage of Object Services and LINQ to Entities</b>	<b>469</b>
<b>Exploring the Generated Entity Classes</b>	<b>470</b>
ModelNameEntities Partial Classes	471
EntityName Partial Classes	472

# Contents

---

Entity Class Serialization	477
Serialization with Deferred-Loaded Associated Entities	478
Serialization with Eager-Loaded Associated Entities	478
<b>Executing eSQL ObjectQueries</b>	<b>480</b>
<b>Enabling Deferred or Eager Loading of Associated Entities</b>	<b>483</b>
Deferred Loading with the Load() Method	484
Eager Loading with Include() Operators	485
<b>Ordering and Filtering Associated EntityCollections during Loading</b>	<b>489</b>
<b>Composing Query Builder Methods to Write ObjectQueries</b>	<b>491</b>
<b>Writing LINQ to Entities Queries</b>	<b>493</b>
Unsupported LINQ Keywords, Standard Query Operators, and Overloads	494
Conventional LINQ to Entities Queries	494
Using the Include() Operator with LINQ to Entities Queries	495
Compiling LINQ to Entity Queries	497
Comparing the Performance of LINQ to Entities Queries	499
<b>Parameterizing Object Queries</b>	<b>500</b>
<b>Summary</b>	<b>500</b>
<b>Chapter 13: Updating Entities and Complex Types</b>	<b>503</b>
<b>Understanding the ObjectContext's ObjectStateManager and Its Children</b>	<b>503</b>
<b>Updating or Deleting Entities</b>	<b>505</b>
Persisting Changes to the Data Store	506
Logging Entities' State	506
Deleting Selected Entities	507
Updating Associated EntitySets or EntityObjects	507
Deleting Entities with Dependent Associated Entities	508
Transacting Updates and Deletions	508
<b>Adding Entities</b>	<b>509</b>
<b>Refreshing Stale Entities</b>	<b>509</b>
<b>Validating Data Additions and Updates</b>	<b>510</b>
<b>Optimizing the ObjectContext Lifetime</b>	<b>513</b>
<b>Comparing the Performance of LINQ to Entities and Out-of-Band SQL Updates</b>	<b>514</b>
<b>Managing Optimistic Concurrency Conflicts</b>	<b>515</b>
Enabling Optimistic Concurrency Management for Entity Properties	516
Implementing Optimistic Concurrency Management with Code	517
<b>Performing CRUD Operations with Stored Procedures</b>	<b>521</b>
Creating the Initial Data Model	521
Adding FunctionImports to Populate the EntitySets	522
Assigning Insert, Update and Delete Stored Procedures to Entities	524
Using a Stored Procedure to Insert a New Entity Instance	524
Updating an Entity Instance and Managing Concurrency with Original Values	525

---

Updating an Entity Instance and Managing Concurrency with a Timestamp Property	527
Deleting Entity Instances with Stored Procedures	528
<b>Working with Complex Types</b>	<b>528</b>
Modeling a Complex Type	529
Modifying the CSDL Section	529
Modifying the MSL Section	530
Reusing a ComplexType Definition	530
<b>Summary</b>	<b>532</b>
<b>Chapter 14: Binding Entities to Data-Aware Controls</b>	<b>533</b>
<b>Binding Windows Form Controls to Entities with Object Data Sources</b>	<b>534</b>
Using the Load(), Include(), and Attach() Methods for Shaping Object Graphs	535
Deferred Loading of Associated Objects with the Load() Method	537
Eager Loading of Multiple Associated Objects with the Include() Method	538
Deferred Loading of Associated Objects with the Attach() Method	539
Selecting the Active Top-Level and Associated Entity Instances	540
Using Unbound ComboBoxes to Specify Associations	543
Adding Event Handlers for ComboBoxColumns	543
Updating Association Sets with the SelectedIndexChanged Event Handler	545
Setting Composite Primary-Key Members with Combo Boxes	546
Detecting Attempts to Change Composite Primary-Key Values	546
Enforcing Object Deletion and Addition Operations for Updates	549
Selecting and Adding a New EntityObject and AssociationSets	551
Persisting Changes to the Data Store	553
<b>Using the EntityDataSource with ASP.NET Server Controls</b>	<b>554</b>
Adding an EntityDataSource Control to an ASP.NET Web Application Page	555
Exploring the Entity Datasource	557
EntityDataSource Properties	557
Where and Group By Clauses	559
EntityDataSource Events	561
Binding and Formatting a GridView Control	562
Using the GroupBy Property and a Drop-Down List to Display Customers by Country	563
Adding a Linked DetailsView	564
<b>Summary</b>	<b>565</b>
<b>Chapter 15: Using the Entity Framework as a Data Source</b>	<b>567</b>
<b>Creating an ADO.NET Data Services Data Source</b>	<b>568</b>
Understanding REST Architecture	570
Creating a Simple Web Service and Consuming It in a Browser	571
Navigating Collections and their Members	573

## Contents

---

Taking Advantage of Query String Options	576
Invoking Service Operations	580
<b>Consuming ADO.NET Data Services with the .NET 3.5 Client Library</b>	<b>581</b>
Executing Queries from Windows Form Clients	583
Executing a Query-String URI from a Client	583
Executing a DataServiceQuery	585
Executing LINQ to REST Queries	586
Returning Associated Entities	587
Executing Batch Queries with DataServiceRequests	589
Changing EntitySets with URI Queries and DataService.SaveChanges()	592
Adding New Entities	592
Updating Entities	594
Deleting Entities	596
Managing Optimistic Concurrency Conflicts	597
Batching Changes	597
<b>Consuming ADO.NET Data Services with the AJAX Client Library</b>	<b>598</b>
Becoming Familiar with JavaScript Object Notation	598
Creating an AJAX Test Client for Astoria	599
Adding, Editing, and Removing Entities	602
<b>Summary</b>	<b>604</b>
<b>Index</b>	<b>607</b>

# Introduction

Language Integrated Query (LINQ), as well as the C# 3.0 and VB 9.0 language extensions to support it, is the most import single new feature of Visual Studio 2008 and the .NET Framework 3.x. LINQ is Microsoft's first attempt to define a universal query language for a diverse set of in-memory collections of generic objects, entities persisted in relational database tables, and element and attributes of XML documents or fragments, as well as a wide variety of other data types, such as RSS and Atom syndication feeds. Microsoft invested millions of dollars in Anders Hejlsberg and his C# design and development groups to add new features to C# 3.0 — such as lambda expressions, anonymous types, and extension methods — specifically to support LINQ Standard Query Operators (SQOs) and query expressions as a part of the language itself.

Corresponding additions to VB 9.0 followed the C# team's lead, but VB's implementation of LINQ to XML offers a remarkable new addition to the language: XML literals. VB's LINQ to XML implementation includes XML literals, which treat well-formed XML documents or fragments as part of the VB language, rather than requiring translation of element and attribute names and values from strings to XML DOM nodes and values.

This book concentrates on hands-on development of practical Windows and Web applications that demonstrate C# and VB programming techniques to bring you up to speed on LINQ technologies. The first half of the book covers SQOs and the concrete implementations of LINQ for querying collections that implement generic `IEnumerable<T>`, `IQueryable<T>`, or both interfaces. The second half is devoted to the ADO.NET Entity Framework, Entity Data Model, Entity SQL (eSQL) and LINQ to Entities. Most code examples emulate real-world data sources, such as the Northwind sample database running on SQL Server 2005 or 2008 Express Edition, and collections derived from its tables. Code examples are C# and VB Windows form or Web site/application projects not, except in the first chapter, simple command-line projects. You can't gain a feel for the behavior or performance of LINQ queries with "Hello World" projects that process arrays of a few integers or a few first and last names.

## Who This Book Is For

This book is intended for experienced .NET developers using C# or VB who want to gain the maximum advantage from the query-processing capabilities of LINQ implementations in Visual Studio 2008 — LINQ to Objects, LINQ to SQL, LINQ to DataSets, and LINQ to XML — as well as the object/relational mapping (O/RM) features of VS 2008 SP1's Entity Framework/Entity Data Model and LINQ to Entities and the increasing number of open-source LINQ implementations by third-party developers.

Basic familiarity with generics and other language features introduced by .NET 2.0, the Visual Studio integrated development environment (IDE), and relational database management systems (RDBMSs), especially Microsoft SQL Server 200x, is assumed. Experience with SQL Server's Transact-SQL (T-SQL) query language and stored procedures will be helpful but is not required. Proficiency with VS 2005, .NET 2.0, C# 2.0, or VB 8.0 will aid your initial understanding of the book's C# 3.0 or VB 9.0 code samples but isn't a prerequisite.

## Introduction

---

*Microsoft's .NET code samples are primarily written in C#. All code samples in this book's chapters and sample projects have C# and VB versions unless they're written in T-SQL or JavaScript.*

## What This Book Covers

Professional ADO.NET 3.5: LINQ and the Entity Framework concentrates on programming the System.Linq and System.Linq.Expressions namespaces for LINQ to Objects, System.Data.Linq for LINQ to SQL, System.Data.Linq for LINQ to DataSet, System.Xml.Linq for LINQ to XML, and System.Data.Entity and System.Web.Entity for EF's Entity SQL.

## How This Book Is Structured

This book is divided into five parts with one to five chapters; each part, except part I, has a four-page introduction that includes an overview and background of the content of the part's chapters. Most chapters build on the knowledge you've gained from preceding chapters. Thus, it's recommended that you work your way through the chapters sequentially. Following is a brief description of each part and its chapters' contents.

### **Part I: Getting a Grip on ADO.NET 3.5**

Part I's sole chapter gives you a hands-on overview of LINQ and its four primary domain-specific implementations: LINQ to SQL, LINQ to DataSets, LINQ to XML, and LINQ to Entities. The chapter's 26 sample projects demonstrate LINQ coding techniques for introductory console, Windows form, and Web site applications in C# 3.0 and VB 9.0.

- **Chapter 1,** "Taking a New Approach to Data Access in ADO.NET 3.5," uses simple C# and VB code examples to demonstrate LINQ to Objects queries against in-memory objects and databinding with LINQ-populated generic List<T> collections, object/relational mapping (O/RM) with LINQ to SQL, joining DataTables with LINQ to DataSets, creating EntitySets with LINQ to Entities, querying and manipulating XML InfoSets with LINQ to XML, and performing queries against strongly typed XML documents with LINQ to XSD. The chapter concludes with a guided tour of the Entity Framework (EF) and Entity Data Model (EDM).

### **Part II: Introducing Language Integrated Query**

The Chapters of Part II back up to explain the enhancements to .NET 3.5 as well as the C# 3.0 and VB 9.0 languages that enable LINQ queries against in-memory objects. LINQ to Objects serves as the foundation for all other concrete LINQ implementations.

- **Chapter 2,** "Understanding LINQ Architecture and Implementation," begins with the namespaces and C# and VB language extensions to support LINQ, LINQ Standard Query Operators (SQOs), expression trees and compiled queries, and a preview of domain-specific implementations. C# and VB sample projects demonstrate object, array, and collection initializers, extension methods, anonymous types, predicates, lambda expressions, and simple query expressions.

- ❑ **Chapter 3**, “Executing LINQ Query Expressions with LINQ to Objects,” classifies the 50 SQOs into operator groups: Restriction, Projection, Partitioning, Join, Concatenation, Ordering, Grouping, Set, Conversion, and Equality, and then lists their keywords in C# and VB. VS 2008 SP1 includes C# and VB versions of the LINQ Project Sample Query Explorer, but the two Explorers don’t use real-world collections as data sources. So, the chapter describes a LINQ in-memory object generator (LIMOG) utility program that writes C# 3.0 or VB 9.0 class declarations for representative business objects that are more complex than those used by the LINQ Project Sample Query Explorers. Sample C# and VB queries with these business objects as data sources are more expressive than those using arrays of a few integers or last names.
- ❑ **Chapter 4**, “Working with Advanced Query Operators and Expressions,” introduces LINQ queries against object graphs with entities that have related (associated) entities. The chapter begins with examples of aggregate operators, explains use of the `Let` temporary local variable operator, shows you how to use `Group By` with aggregate queries, conduct the equivalent of left outer joins, and take advantage of the `Contains()` SQO to emulate SQL’s `IN()` function. You learn how to compile queries for improved performance, and create mock object classes for testing without the overhead of queries against relational persistence stores.

## **Part III: Applying Domain-Specific LINQ Implementations**

Part III’s chapters take you to the next level of LINQ applications, domain-specific versions for SQL Server 200x databases, working with `DataTable`s, managing XML Infosets, and a few members of the rapidly expanding array of third-party LINQ implementations. Each chapter offers C# and VB sample projects to complement the chapter’s content.

- ❑ **Chapter 5**, “Using LINQ to SQL and the `LinqDataSource`,” introduces LINQ to SQL as Microsoft’s first O/RM tool to reach released products status and shows you how to autogenerate class files for entity types with the graphical O/R Designer or command-line `SqlMetal.exe`. The chapter also explains how to edit `*.dbml` mapping files in the Designer or XML Editor, instantiate `DataContext` objects, and use LINQ to SQL as a Data Access Layer (DAL) with T-SQL queries or stored procedures. The chapter closes with a tutorial for using the ASP.NET `LinqDataSource` control with Web sites or applications.
- ❑ **Chapter 6**, “Querying `DataTable`s with LINQ to `DataSet`s,” begins with a comparison of `DataSet` and `DataContext` objects and features, followed by a description of the `DataSetExtensions`. Next comes querying untyped and typed `DataSet`s, creating lookup lists, and generating `LinqDataViews` for databinding with the `AsDataView()` method. The chapter ends with a tutorial that shows you how to copy LINQ query results to `DataTable`s.
- ❑ **Chapter 7**, “Manipulating Documents with LINQ to XML,” describes one of LINQ’s most powerful capabilities: managing XML Infosets. The chapter demonstrates that LINQ to XML has query and navigation capabilities that equal or surpass XQuery 1.0 and XPath 2.0. It also shows LINQ to XML document transformation can replace XQuery and XSLT 1.0+ in the majority of common use cases. You learn how to use VB 9.0’s XML literals to construct XML documents, use `GroupJoin()` to produce hierarchical documents, and work with XML namespaces in C# and VB.
- ❑ **Chapter 8**, “Exploring Third-Party and Emerging LINQ Implementations,” describes Microsoft’s Parallel LINQ (also called PLINQ) for taking advantage of multiple CPU cores in LINQ to Objects queries, LINQ to REST for translating LINQ queries into Representational State Transfer URLs that define requests to a Web service with the HTML GET, POST, PUT, and DELETE methods, and Bart de Smet’s LINQ to Active Directory and LINQ to SharePoint third-party implementations.

## Part IV: Introducing the ADO.NET Entity Framework

Part IV gives you insight to the architecture of the EDM and its first concrete implementation as an O/RM by EF. Microsoft intends the EDM to become a universal specification for defining a conceptual model of data expressed as *entities* and *associations* (also called *relationships*) across a wide variety of data sources. In addition to relational databases used for object persistence, the EDM is expected to be a future component of synchronization and reporting services.

- ❑ **Chapter 9**, “Raising the Level of Data Abstraction with the Entity Data Model,” starts with a guided tour of the development of EDM and EF as an O/RM tool and heir apparent to ADO.NET DataSets, provides a brief description of the entity-relationship (E-R) data model and diagrams, and then delivers a detailed analysis of EF architecture. Next comes an introduction to the Entity SQL (eSQL) language, eSQL queries, client views, and Object Services, including the `ObjectContext`, `MetadataWorkspace`, and `ObjectStateManager`. Later chapters describe eSQL and these objects in greater detail. Two C# and VB sample projects expand on the chapter’s eSQL query and Object Services sample code.
- ❑ **Chapter 10**, “Defining Conceptual, Mapping, and Storage Schema Layers,” provides detailed insight into the structure of the `*.edmx` file that generates the `*.ssdl` (storage schema data language), `*.msl` (mapping schema language), and `*.csdl` files at runtime. You learn how to edit the `*.edmx` file manually to accommodate modifications that the graphic EDM Designer can’t handle. You learn how to implement the Table-per-Hierarchy (TPH) inheritance model and traverse the `MetadataWorkspace` to obtain property values. Four C# and VB sample projects demonstrate mapping, substituting stored procedures for queries, and TPH inheritance.
- ❑ **Chapter 11**, “Introducing Entity SQL,” examines EF’s new eSQL dialect that adds keywords to address the differences between querying entities and relational tables. You learn to use Zlatko Michaelov’s eBlast utility to write and analyze eSQL queries, then dig into differences between eSQL and T-SQL `SELECT` queries. (eSQL v1 doesn’t support `INSERT`, `UPDATE`, `DELETE` and other SQL Data Manipulation Language constructs.) You execute eSQL queries against the `EntityClient`, measure the performance hit of eSQL compared to T-SQL, execute parameterize eSQL queries, and use SQL Server Compact 3.5 as a data store. C# and VB Sample projects demonstrate the programming techniques that the chapter covers.

## Part V: Implementing the ADO.NET Entity Framework

Part V covers the practical side of implementing EF as a O/RM with SQL Server 200x, working with LINQ to Entities queries, using LINQ to Entities to update conventional entities and those with complex types, databinding Object Services’ `ObjectContext` to Windows form and ASP.NET server controls, and taking advantage of EF as a data source for other data-aware .NET technologies. Each chapter includes C# and VB sample code to enhance the chapters’ code examples.

- ❑ **Chapter 12**, “Taking Advantage of Object Services and LINQ to Entities,” concentrates manipulating the Object Services API’s `ObjectContext`. The chapter continues with demonstrating use of partial classes for the `ModelNameEntities` and `EntityName` objects, executing eSQL `ObjectQuery`s, and deferred or eager loading of associated entities, including ordering and filtering the associated entities. Instructions for composing `QueryBuilder` methods for `ObjectQuery`s, LINQ to Entities queries, and parameterizing `ObjectQuery`s complete the chapter.

- ❑ **Chapter 13**, “Updating Entities and Complex Types,” shows you how to perform create, update, and delete (CUD) operations on EntitySets and manage optimistic concurrency conflicts. It starts with a detailed description of the `ObjectContext.ObjectStateManager` and its child objects, which perform object identification and change tracking operations with `EntityKeys`. The chapter also covers validation of create and update operations, optimizing the `DataContext` lifetime, performing updates with stored procedures, and working with complex types.
- ❑ **Chapter 14**, “Binding Data Controls to the ObjectContext,” describes creating design-time data sources from `ObjectContext.EntitySet` instances, drag-and-drop addition of `BindingNavigator`, `BindingSource`, bound `TextBox`, and `DataGridView` controls to Windows forms. You also learn how to update `EntityReference` and `EntitySet` values with `ComboBox` columns in `DataGridView` controls. (You can’t update `EntitySet` values directly; you must delete and add a new member having the required value.) The chapter concludes with a demonstration of the use of the ASP.NET `EntityDataSource` control bound to `GridView` and `DropDownList` controls.
- ❑ **Chapter 15**, “Using the Entity Framework As a Data Source,” concentrates on using EF as a data source for the ADO.NET Data Services Framework (the former codename “Project Astoria” remains in common use), which is the preferred method for deploying EF v1 as a Web service provider. (EF v2 is expected to be able to support *n*-tier data access with Windows Communication Foundation [WCF] directly). A Windows form example uses Astoria’s .NET 3.5 Client Library to display and update entity instances with the Atom Publication (AtomPub or APP) wire format. The Web form project uses the AJAX Client Library and JavaScript Object Notation (JSON) as the wire format.

## Conventions

To help you get the most from the text and keep track of what’s happening, we’ve used a number of conventions throughout the book.

*Notes, tips, hints, tricks, and asides to the current discussion are offset and placed in italics like this.*

As for styles in the text:

- ❑ We *highlight* new terms and important words when we introduce them.
- ❑ We show keyboard strokes like this: `Ctrl+A`.
- ❑ We show filenames, URLs, and code within the text like so: `persistence.properties`.
- ❑ We present code in two different ways:

We use a monofont type with no highlighting for most code examples.

We use gray highlighting to emphasize code that’s particularly important in the present context.

# Source Code

As you work through the examples in this book, you may choose either to type in all the code manually or to use the source code files that accompany the book. All of the source code used in this book is available for downloading at [www.wrox.com](http://www.wrox.com). Once at the site, simply locate the book's title (either by using the Search box or by using one of the title lists) and click the Download Code link on the book's detail page to obtain all the source code for the book.

*Because many books have similar titles, you may find it easiest to search by ISBN; this book's ISBN is 978-0-470-18261-1.*

Once you download the code, just decompress it with your favorite compression tool. Alternately, you can go to the main Wrox code download page at [www.wrox.com/dynamic/books/download.aspx](http://www.wrox.com/dynamic/books/download.aspx) to see the code available for this book and all other Wrox books.

# Errata

We make every effort to ensure that there are no errors in the text or in the code. However, no one is perfect, and mistakes do occur. If you find an error in one of our books, like a spelling mistake or faulty piece of code, we would be very grateful for your feedback. By sending in errata you may save another reader hours of frustration and at the same time you will be helping us provide even higher-quality information.

To find the errata page for this book, go to [www.wrox.com](http://www.wrox.com) and locate the title using the Search box or one of the title lists. Then, on the book details page, click the Book Errata link. On this page you can view all errata that has been submitted for this book and posted by Wrox editors. A complete book list including links to each book's errata is also available at [www.wrox.com/misc-pages/booklist.shtml](http://www.wrox.com/misc-pages/booklist.shtml).

If you don't spot "your" error on the Book Errata page, go to [www.wrox.com/contact/techsupport.shtml](http://www.wrox.com/contact/techsupport.shtml) and complete the form there to send us the error you have found. We'll check the information and, if appropriate, post a message to the book's errata page and fix the problem in subsequent editions of the book.

# p2p.wrox.com

For author and peer discussion, join the P2P forums at [p2p.wrox.com](http://p2p.wrox.com). The forums are a Web-based system for you to post messages relating to Wrox books and related technologies and interact with other readers and technology users. The forums offer a subscription feature to e-mail you topics of interest of your choosing when new posts are made to the forums. Wrox authors, editors, other industry experts, and your fellow readers are present on these forums.

At <http://p2p.wrox.com> you will find a number of different forums that will help you not only as you read this book but also as you develop your own applications. To join the forums, just follow these steps:

- 1.** Go to [p2p.wrox.com](http://p2p.wrox.com) and click the Register link.
- 2.** Read the terms of use and click Agree.

- 3.** Complete the required information to join as well as any optional information you wish to provide, and click Submit.
- 4.** You will receive an e-mail with information describing how to verify your account and complete the joining process.

*You can read messages in the forums without joining P2P but in order to post your own messages, you must join.*

Once you join, you can post new messages and respond to messages other users post. You can read messages at any time on the Web. If you would like to have new messages from a particular forum e-mailed to you, click the Subscribe to this Forum icon by the forum name in the forum listing.

For more information about how to use the Wrox P2P, be sure to read the P2P FAQs for answers to questions about how the forum software works as well as many common questions specific to P2P and Wrox books. To read the FAQs, click the FAQ link on any P2P page.



**Professional**

**ADO.NET 3.5 with LINQ and the Entity Framework**



# **Part I : Getting a Grip on ADO.NET 3.5**

**Chapter 1:** Taking a New Approach to Data Access in ADO.NET 3.5



# Taking a New Approach to Data Access in ADO.NET 3.5

Visual Studio (VS) 2008 SP1 is a major upgrade to Visual Studio 2005 and the Visual Basic 8.0 and C# 2.0 languages. ADO.NET 3.5 introduces many new language and data access features to ADO.NET 2.0 and its DataSet-oriented programming model. The primary objectives of these additions are to:

- ❑ Increase programmer productivity by reducing the amount of code required to complete a particular data-related task, such as querying and updating relational data sources, DataSets, or XML documents
- ❑ Enable strong type-checking of query syntax for relational data and query resultsets
- ❑ Reduce or eliminate what's often called the *impedance mismatch* between the code for managing data, which usually involves Structured Query Language (SQL), and object-oriented (OO) programming
- ❑ Provide data-intensive Windows form projects with a local, synchronizable data cache derived from SQL Server Compact Edition v3.5
- ❑ Enhance DataSet integration with *n*-tier architecture

VS attempts to accomplish the preceding objectives with the following new LINQ language extensons to C# and VB, domain-specific implementations, and LINQ-related components or frameworks:

- ❑ **Language Integrated Query (LINQ)** extensions to the Visual Basic 9.0 and C# 3.0 languages and compilers provide developers with a set of standard query operators to query a variety of data sources with a standard query language similar to SQL. LINQ extensions to VB and C# are said to implement the *LINQ pattern*.
- ❑ **LINQ to Objects** is the default LINQ implementation, which enables developers to express and execute queries against in-memory collections that implement the `IEnumerable` or `IQueryable` interface.

## Part I: Getting a Grip on ADO.NET 3.5

---

- ❑ **LINQ to SQL** (formerly DLinq) extensions to LINQ to Objects provide a basic object/relational mapping (O/RM) tool to generate and manage entity classes for business objects, and persist entity instances in SQL Server 200x tables.
- ❑ **LinqDataSource** is an ASP.NET 3.5 server control for simplifying binding other data-enabled Web controls to business objects created with LINQ to SQL.

*In OO-speak, to persist means to save in-memory objects to and load them from a data store, which can be a relational, object or flat-file database, or an Extensible Markup Language (XML) or other type of file.*

- ❑ **LINQ to DataSets** is an extension that enables LINQ to execute queries with joins over DataTables of typed or untyped ADO.NET DataSets.
- ❑ **LINQ to XML** (formerly XLinq) extensions to LINQ to Objects enable XML document processing with standard LINQ queries and constructing XML documents with a declarative syntax.
- ❑ **Entity Framework (EF)**, Entity Data Model (EDM), EntityClient .NET data provider for SQL Server 200x, Entity SQL (eSQL) query language, and related designers, wizards, and tools map a data store's physical implementation to a conceptual schema. The conceptual schema supports an upper layer of custom business objects, which can represent a domain model.
- ❑ **LINQ to Entities** is a LINQ implantation that generates eSQL to enable querying the Entity Framework's Object Services layer with LINQ syntax.
- ❑ **EntityDataSource** corresponds to a LinqDataSource control for business objects generated by EF's EDM.

Following are a few of the most important new related technologies, frameworks, and implementations that take advantage of or enhance LINQ:

- ❑ **LINQ to XSD** extensions to LINQ to XML deliver strong data-typing to LINQ to XML queries by reference to a formal or inferred XML schema for the source document. LINQ to XSD was an unsupported alpha version of an incubator project when this book was written.
- ❑ **ADO.NET Data Services Framework** (formerly Project Astoria) enables developers to make data available as a simple Representational State Transfer (REST) Web service with Atom Publication (AtomPub or APP) format or JavaScript Object Notation (JSON) as the wire format. EF is Astoria's preferred data source; LINQ to SQL or other LINQ-enabled data sources create read-only Astoria services.
- ❑ **ASP.NET Dynamic Data (DD)** is a framework for autogenerating complete data-intensive Web sites from a LINQ to SQL or LINQ to Entities data source. DD is a highly enhanced version of an earlier project named BLINQ. DD can create an administrative Web site for a complex database in a few minutes by a process known as scaffolding. Ruby on Rails introduced Web developers to automated scaffolding.
- ❑ **Windows Communication Framework (WCF) Service templates** simplify development of n-tier service-oriented applications that isolate the data-access tier from business object and presentation tiers by interposing a service layer. Astoria implements a string-based subset of LINQ query syntax. Astoria is an example of a LINQ-oriented framework that takes advantage of .NET 3.5 SP1 WCF templates.

## Chapter 1: Taking a New Approach to Data Access in ADO.NET 3.5

---

- ❑ **Parallel LINQ** (PLINQ) is a set of extensions to LINQ to Objects that enable parallel processing of LINQ queries on multi-core PCs. PLINQ was in the Community Technical Preview (CTP) stage when this book was written.
- ❑ **Hierarchical updates to typed DataSets** with the `TableAdapterManager.UpdateAll()` method greatly reduce the amount of code required to update multiple related tables in the underlying database.
- ❑ **SQL Server Compact** (SSCE) 3.5 enables local data caching with a lightweight in-process relational database management system. LINQ to SQL and EF can use SSCE 3.5 as a data source but prevent use of the graphic O/RM tools to edit the resulting entities.
- ❑ **Microsoft Synchronization Framework** components implement occasionally connected systems (OCS) by synchronizing SSCE caches with SQL Server 200x databases.

The preceding list includes substantially more new data-related technologies and components than delivered by any .NET Framework and Visual Studio version since 1.0; this book covers all but the last four items. Examples demonstrate WCF service-oriented, *n*-tier architectures for Astoria projects that use LINQ to SQL or LINQ to Entities their data source. However, this book doesn't attempt to cover other WCF features.

This chapter is an introduction to the two principal topics of this book: LINQ and the ADO.NET Entity framework.

*Microsoft's Server and Tools Division dropped EF and its components from the release-to-manufacturing (RTM) version Visual Studio 2008 on April 29, 2007. The ADO.NET Team subsequently released EF CTP and Beta editions with the intention of adding it to VS 2008 and .NET 3.5 in the first half of 2008. EF and EDM released with .NET 3.5 SP1 and VS 2008 SP1 on August 11, 2008. This book was written with the RTM versions of .NET 3.5 SP1 and VS 2008 SP1.*

## Language Integrated Query (LINQ)

LINQ is a revolutionary programming methodology that transforms the relationship between programs and data. LINQ defines a .NET application programming interface (API) and set of extensions to the Visual Basic and C# languages that enables querying diverse data types with a single syntax that's similar to Structured Query Language (SQL). Writing queries in your .NET language of choice enables strong typing and support for programmer productivity enhancing features, such as statement completion and IntelliSense.

The LINQ framework permits custom extensions to support common data types or exotic data domains. When this book was written, the .NET Framework and Visual Studio 2008 supported LINQ implementations for querying collections of in-memory objects (LINQ to Objects), tables of SQL Server 200x databases (LINQ to SQL), ADO.NET DataSets (LINQ to DataSets), XML Infosets (LINQ to XML), and business entities created by the ADO.NET Entity Framework. The relative ease with which developers can write LINQ data providers has spawned a cottage industry of independent programmers who write LINQ data providers for recreation, demonstrations, presentations, or simply to demonstrate their C# chops.

## Part I: Getting a Grip on ADO.NET 3.5

---

You construct LINQ queries over .NET collections with statements composed of C# 3.0 or VB 9.0 keywords (called *LINQ Standard Query Operators* — SQO — or *standard sequence operators*), many of which have corresponding or similar SQL reserved words. The `System.Linq` namespace's `Enumerable` class defines the standard query operators. The .NET C# 3.0 and VB 9.0 compilers transform the query statements to the Common Language Runtime's (CLR's) intermediate language (IL) for execution. Therefore, the LINQ programming model is a first-class member of the .NET 3.5 Framework; it is *not an add-in*.

Here's the general form of a basic LINQ query, as defined by Visual Studio 2008's statement completion feature:

### C# 3.0

```
var query = from element in collection[, from element2 in collection2]
            where condition _
            orderby orderExpression [ascending|descending][, orderExpression2 ...]
            select [alias = ]columnExpression[, alias2 = ]columnExpression2]
```

### VB 9.0

```
Dim Query = From Element In Collection[,Element2 In Collection2] _
            Where Condition _
            Order By OrderExpression [Ascending|Descending][, _
            OrderExpression2 ...] _
            Select [Alias = ]ColumnExpression[, Alias2 = ]ColumnExpression2
```

*This book uses C# lower-case and VB PascalCase interchangeably, except where keywords differ between the two languages, such as `orderby` and `Order By` in the preceding example. Preference is given to PascalCase to emphasize that the words are keywords.*

The `Dim` and `var` keywords specify *local variable type inference* for the variable `Query/query`. Local variable type inference (LVTI) is a new C# 3.0/VB 9.0 language feature. VB's new `Option Infer {On|Off}` statement controls LVTI, and the default for new projects is `On`, which enables LVTI with `Option Strict On`.

The preceding two examples correspond to this elementary SQL query syntax:

### SQL

```
SELECT [Element.ColumnExpression[, Element2.ColumnExpression2]]
FROM Collection AS Element[, Collection2 AS Element2]
WHERE Condition
ORDER BY OrderExpression [ASCENDING|DESCENDING][, OrderExpression2 ...]
```

The principal difference between LINQ and SQL queries is the position of the LINQ `From` clause, which determines the source of the data, at the start of the query. `From` before `Select` is necessary to enable IntelliSense. (SQL compilers perform this rearrangement before compiling the query.)

*If you're familiar with XQuery's FLWOR (For, Let, Where, Order By, Return) syntax, you'll notice the similarity of the keywords and clause sequences. XQuery's `for...in` keywords correspond to LINQ's `From...In`; C#'s LINQ implementation and XQuery use `Let/let` for variable assignment. XQuery's `return` corresponds to LINQ's `Select`.*

## Chapter 1: Taking a New Approach to Data Access in ADO.NET 3.5

---

LINQ, SQL, and XQuery are *declarative* programming languages. In contrast, C#, VB, Java, and Fortran are *imperative* languages. Declarative languages describe the programmer's goal, while imperative programs describe a specific series of steps or algorithms to achieve the goal. LINQ's use of a declarative query language enables the LINQ API and its extensions to optimize query processing at runtime in the same manner as SQL query optimizers.

You can query any information source that exposes the `IEnumerable<T>` or `IQueryable<T>` interface. Almost all .NET collections implement `IEnumerable<T>`, which .NET 2.0 introduced. `IQueryable<T>` is a new, enhanced version of `IEnumerable<T>` that enables dynamic queries and other query features.

*Chapter 2 includes an introduction to the standard query operators, as well as the `IEnumerable<T>` and `IQueryable<T>` interfaces.*

Third parties can create their own LINQ implementations by exposing the `IEnumerable<T>` interface for their collection(s) with extension methods. Following in alphabetical order are some of the more imaginative third-party LINQ implementations that were available when this book was written:

- ❑ **LINQ to Amazon** by Fabrice Marguerie was the first generally available third-party LINQ provider. LINQ to Amazon returns lists of books meeting specific criteria.
- ❑ **LINQ to Expressions** (MetaLinq) by Aaron Erickson (the developer of Indexes for Objects, i4o) lets you query over and edit expression trees with LINQ. Like .NET strings, LINQ expression trees are immutable; LINQ to Expressions changes a LINQ expression by making a copy, modifying the copy, and then replacing the original.
- ❑ **LINQ to Flickr** by Mohammed Hossam El-Din uses the open-source FlickrNet C# library for digital images as its infrastructure.
- ❑ **LINQ to LDAP** by Bart de Smet is a “query provider for LINQ that’s capable of talking to Active Directory (and other LDAP data sources potentially) over LDAP.”
- ❑ **LINQ to NHibernate** by Ayende Rahien (a.k.a. Oren Eini) translates LINQ queries to NHibernate Criteria Queries and was in the process of updating to .NET 3.5 when this book was written.
- ❑ **LINQ4SP** by L&M Solutions of Hungary is an implementation of LINQ for SharePoint lists by an independent SharePoint specialist.
- ❑ **LINQ to Streams** by Oren Novotny, this book’s technical editor, processes continuous data streams, such as stock tickers, network monitors, and sensor data.

*Chapter 8 describes several of the preceding third-party LINQ flavors, and other Microsoft incubator implementations. The OakLeaf Systems blog provides a permanent, up-to-date list of third-party LINQ implementations at <http://oakleafblog.blogspot.com/2007/03/third-party-linq-providers.html>.*

The LINQ standard query operators apply to all implementations, so developers don’t need to learn new query languages or dialects, such as SQL for relational databases, eSQL for the Entity Framework and EDM, XPath, or XQuery for XML Infosets, Lightweight Directory Access Protocol (LDAP) to query Active Directory and other LDAP-based directory systems, Collaborative Application Markup Language for SharePoint, and HibernateSQL (HSQL) for NHibernate.

## Part I: Getting a Grip on ADO.NET 3.5

The sections that follow briefly describe Microsoft's LINQ implementations for the release version of Visual Studio 2008, which are illustrated by Figure 1-1. The ADO.NET team owns the LINQ to ADO.NET group members, which includes LINQ to Entities, LINQ to DataSets, and LINQ to SQL. The C# group owns the LINQ specification and LINQ to Objects, the ASP.NET team owns the LinqDataSource and EntityDataSource, and the XML Data Programmability team is responsible for LINQ to SQL and EF/EDM. The XML team handles LINQ to XML and is indirectly responsible for LINQ to XSD.

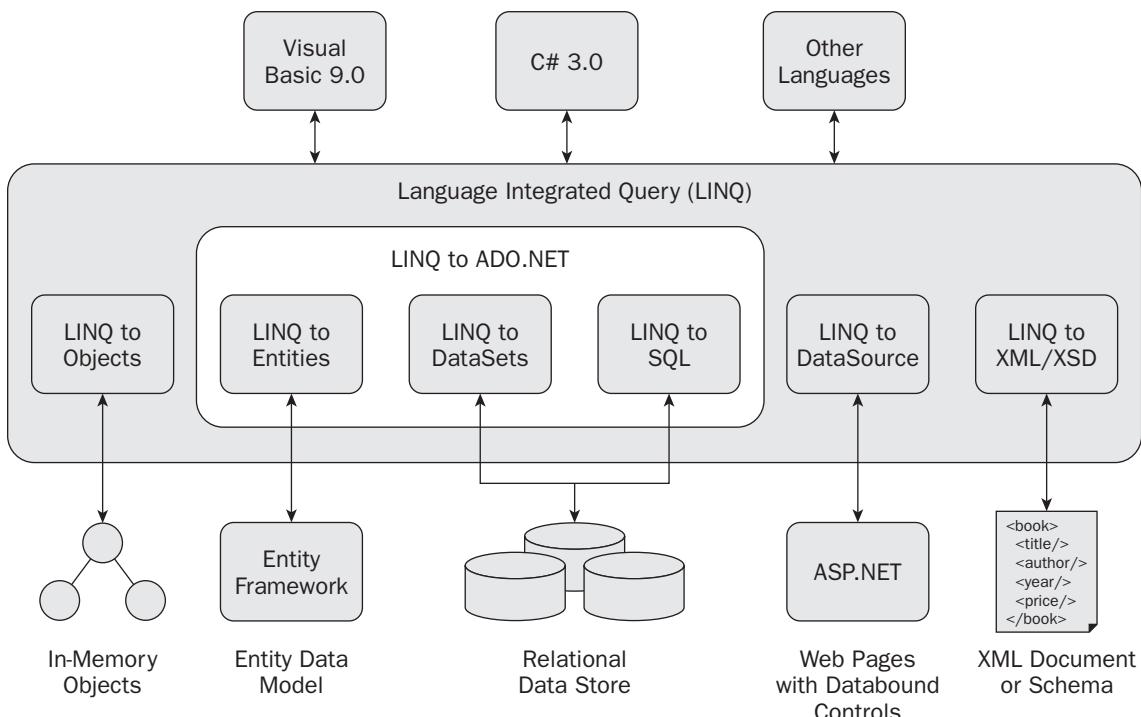


Figure 1-1

The chapters of Part II, “Introducing Language Integrated Query,” provides detailed coverage of extensions to the C# 3.0 and VB 9.0 languages to support LINQ and shows you the inner workings of LINQ to Objects. Part III, “Applying Domain-Specific LINQ Implementations,” covers LINQ to SQL, the ASP.NET LinqDataSource, LINQ to DataSets, and LINQ to XML/LINQ to XSD. Chapter 12 explains the differences between LINQ to Entities and LINQ to SQL.

### LINQ to Objects

LINQ to Objects is Microsoft's default LINQ implementation, which is included in the .NET Framework 3.5's System.Core.dll library. This means that it also is part of the Dynamic Language Runtime (DLR) that Microsoft announced at the MIX07 conference at the end of April 2007. The DLR release version will support JavaScript, IronPython, IronRuby, and the next VB version, VB 10.0 but currently called VBx. Silverlight 2 uses System.Core.dll to support LINQ and also supports LINQ to XML.

According to Paul Vick, the author of the VB.NET language specification, “Part of VBx is a hostable managed component (written in Visual Basic, no less!) built on top of the DLR. Since Silverlight can host DLR languages, this enables you to send Visual Basic code to a Silverlight instance and have it dynamically compiled and run.” The XML Data Programmability team plans to support LINQ to XML in the DLR under Silverlight. VBx currently is the only DLR language that will support LINQ or LINQ to XML, but you can mix VBx source code with that of other dynamic languages.

## Sample LINQ Queries

Following is the C# 3.0 version of a LINQ query (\WROX\ADONET\Chapter01\CS\LINQforProcesses\LINQforProcessesCS.sln) that lists the name, memory consumption of the working set in bytes, and the number of threads for all processes running on your computer that have spawned more than 10 threads. The `orderby` clause sorts the list in descending order of memory consumed. VS automatically formats the LINQ query as you type it.

### C# 3.0

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;

namespace LINQforProcessesCS
{
    class LINQConsole
    {
        static void Main(string[] args)
        {
            var procs = from proc in Process.GetProcesses()
                        where proc.Threads.Count > 10
                        orderby proc.WorkingSet64 descending
                        select new { proc.ProcessName, Memory = proc.WorkingSet64,
                                    Threads = proc.Threads.Count };

            foreach (var proc in procs)
                Console.WriteLine(proc);
            Console.ReadLine();
        }
    }
}
```

When you create a C# console project, the Console Application template adds the first five `using` directives, which include `using System.Linq`. The `System.Diagnostics.Process` `.GetProcesses()` method has many properties so specifying `select proc`, which would write out all property values, isn’t very useful. Specifying a *projection* (specific field list) rather than the `proc` object (the equivalent of SQL’s `*`) requires adding a new constructor to `select`. The `select` projection aliases `WorkingSet64` with `Memory` and `Count` with `Threads` to produce the following console output:

```
{ ProcessName = devenv, Memory = 72781824, Threads = 21 }
{ ProcessName = devenv, Memory = 64450560, Threads = 23 }
{ ProcessName = svchost, Memory = 51621888, Threads = 35 }
{ ProcessName = explorer, Memory = 31342592, Threads = 31 }
```

## Part I: Getting a Grip on ADO.NET 3.5

```
{ ProcessName = LINQforProcessesCS.vshost, Memory = 21598208, Threads = 13 }
...
{ ProcessName = taskeng, Memory = 8331264, Threads = 16 }
{ ProcessName = sqlservr, Memory = 5091328, Threads = 24 }
{ ProcessName = System, Memory = 2818048, Threads = 108 }
{ ProcessName = lsass, Memory = 1589248, Threads = 14 }
```

Figure 1-2 shows Visual Studio 2008's C# IntelliSense output for the query (procs) object.

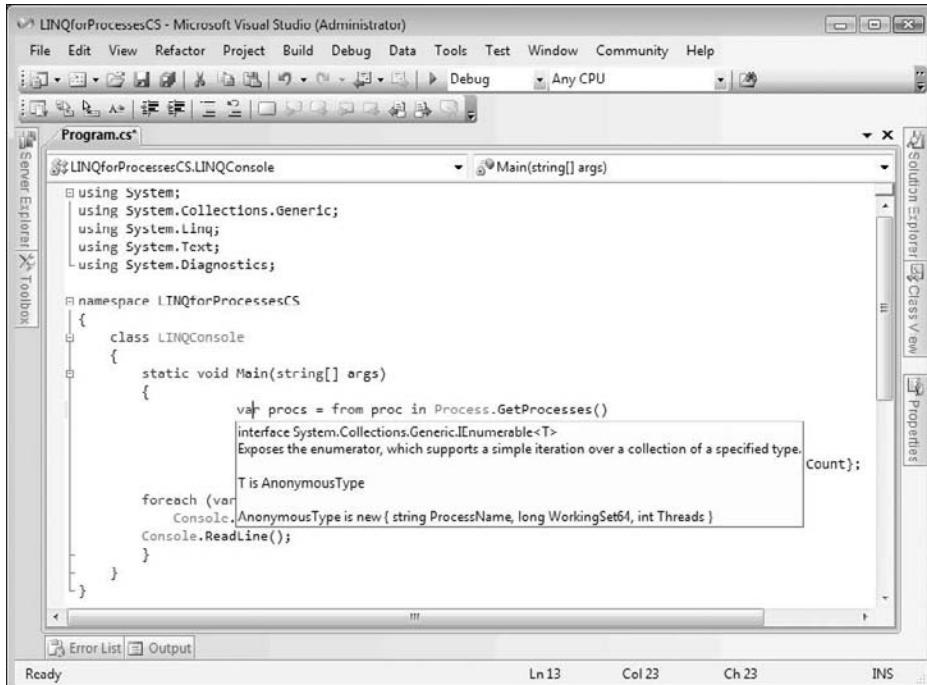


Figure 1-2

Following is the VB 9.0 version (\WROX\ADONET\Chapter01\VB\LINQforProcesses\LINQforProcessesVB.sln) of the preceding query. The VB Console Application template doesn't add Imports directives for you, so you must add the Imports System.Linq directive manually. Setting UseSimpleSelect = False formats the output to duplicate that of the C# query.

## VB 9.0

```
Option Explicit On
Option Strict On
Option Infer On

Imports System.Linq
Imports System.Diagnostics

Module Module1
    Sub Main()
        Dim UseSimpleSelect = False
        If UseSimpleSelect Then
            'Outputs "WorkingSet64 = " and "Count = "
            Dim Procs = From Proc In Process.GetProcesses _
                        Where Proc.Threads.Count > 10 _
                        Order By Proc.WorkingSet64 Descending _
                        Select Proc.ProcessName, Proc.WorkingSet64, Proc.Threads.Count
            For Each Proc In Procs
                Console.WriteLine(Proc)
            Next
        Else
            'Outputs "Memory = " and "Threads = "
            Dim Procs = From Proc In Process.GetProcesses _
                        Where Proc.Threads.Count > 9 _
                        Order By Proc.WorkingSet64 Descending _
                        Select New With {Proc.ProcessName, .Memory = Proc.WorkingSet64, _
                                         .Threads = Proc.Threads.Count}
            For Each Proc In Procs
                Console.WriteLine(Proc)
            Next
        End If
        Console.ReadLine()
    End Sub
End Module
```

VB's query syntax differs from the C# flavor primarily by keyword proper capitalization, but `Order By` (with a space) replaces `orderby`. VB doesn't require the `New` constructor unless you add aliases, such as `Memory` and `Threads`, in which case you must add the VB-specific `New With` constructor for the `Select` projection.

VB's statement completion and IntelliSense for LINQ queries differs dramatically from C#'s, as illustrated by Figure 1-3. This figure is a multiple exposure to demonstrate LINQ query statement completion with the common keywords shown in the list box on the left, and IntelliSense for the keywords within the query code.

## Part I: Getting a Grip on ADO.NET 3.5

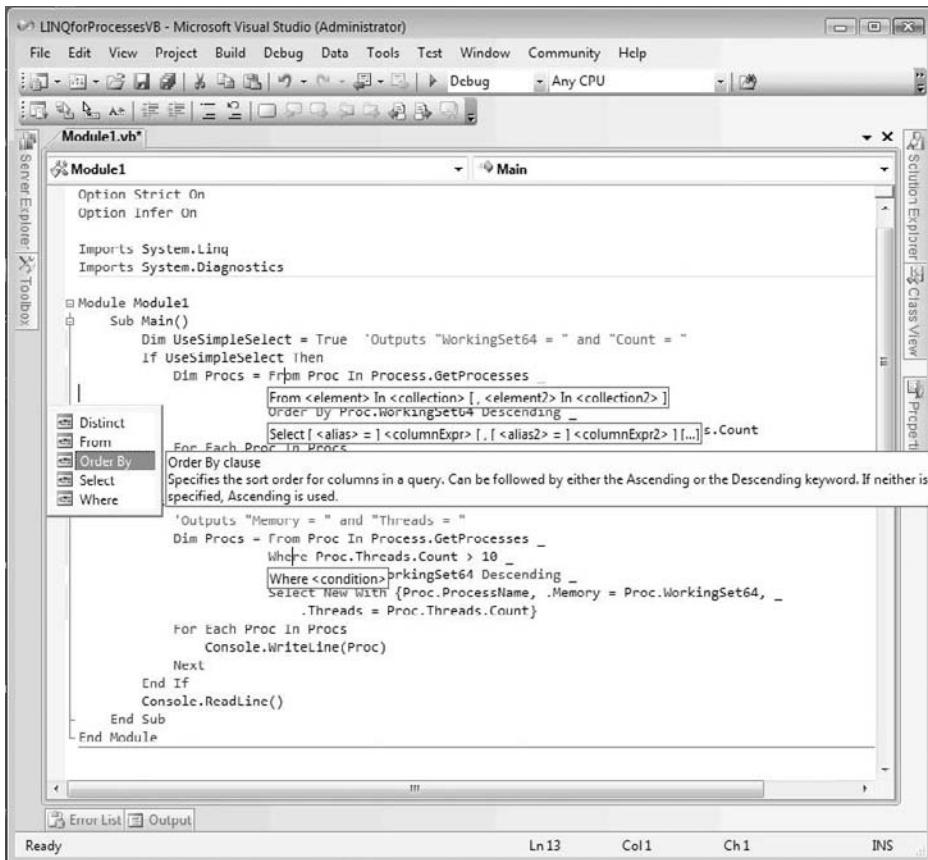


Figure 1-3

### LINQ Data Binding with the `ToList()` Operator

One of the common uses of LINQ queries is as a relatively lightweight data source for databound controls. You can change the data type of the query variable from the default `IEnumerable<T>` to a generic `Array<T>, Dictionary<T>, List<T>, or Lookup<T>`, where `<T>` is an Anonymous Type, with the `ToDictionary()`, `ToList()`, or `ToLookup()` operator, respectively. A `ToBindingList()` operator was present in early CTPs, but it was removed from Visual Studio Orcas Beta1 with the recommendation that the query variable itself serve as the data source. This works for some databound controls but not for ASP.NET paged DataGrid controls. Applying the `ToList()` operator adds support for DataGrid paging.

## Chapter 1: Taking a New Approach to Data Access in ADO.NET 3.5

---

ASP.NET's *LinqDataSource*, which closely resembles the *SqlDataSource*, supports all features of the *SqlDataSource*, including paging and create, retrieve, update and delete (CRUD) operations. For more information see this chapter's "LinqDataSource" topic or Chapter 5.

The following C# code behind the *Processes.aspx* page of the sample file system WebSite project (\WROX\ADONET\Chapter01\CS\LINQforBinding\LINQforBindingCS.sln) applies the *ToList()* operator to the LINQ query, designates the *List<T>* as the DataGrid's *DataSource* property, and then invokes the *DataBind* method to display the two pages of process data. As was the case for console apps, the starting codebehind class for *Default.aspx* pages includes a *using System.Linq* directive.

### C# 3.0

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Diagnostics;

public partial class _Processes : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        var procs = (from proc in Process.GetProcesses()
                     where proc.Threads.Count > 10
                     orderby proc.WorkingSet64 descending
                     select new
                     {
                         proc.ProcessName,
                         Memory = proc.WorkingSet64,
                         Threads = proc.Threads.Count
                     }).ToList();
        ProcessesGridView.DataSource = procs;
        ProcessesGridView.DataBind();
    }
}
```

Figure 1-4 shows the populated DataGrid.

## Part I: Getting a Grip on ADO.NET 3.5

ProcessName	Memory	Threads
devenv	180686848	26
devenv	158617600	38
svchost	45273088	35
explorer	35770368	37
svchost	16375808	46
SearchIndexer	11370496	14
svchost	9953280	29
svchost	9682944	14
svchost	8257536	32
taskeng	8073216	16

Figure 1-4

Here's the VB code in the `Processes.aspx.vb` file of the VB version of `Processes.aspx` page.

### VB 9.0

```
Option Explicit On
Option Strict On
Option Infer On

Imports System.Linq
Imports System.Diagnostics

Partial Class _Processes
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load
        Dim Procs = (From Proc In Process.GetProcesses _
            Where Proc.Threads.Count > 9 _
            Order By Proc.WorkingSet64 Descending _
            Select New With {Proc.ProcessName, .Memory = Proc.WorkingSet64, _
                .Threads = Proc.Threads.Count}).ToList
        Me.ProcessesGridView.DataSource = Procs
        Me.ProcessesGridView.DataBind()
    End Sub
End Class
```

## Chapter 1: Taking a New Approach to Data Access in ADO.NET 3.5

---

The LINQforProcesses and LINQforBinding applications and their sample code demonstrate that substituting LINQ queries and their set manipulation techniques for conventional procedural code that operates on collections can save developers a substantial amount of planning and coding time.

*There's an additional pair of sample projects (\WROX\ADONET\Chapter01\CS \LINQforArrays\LINQforArraysCS.sln and \WROX\ADONET\Chapter01\VB \LINQforArrays\LINQforArraysVB.sln) to demonstrate similar LINQ queries on arrays of strings.*

### **LINQ to SQL**

LINQ to SQL is more than just a LINQ implementation for relational databases; it includes an easy-to-use graphical object/relational mapping (O/RM) tool. The O/RM tool generates an entity class for each table you select from the SQL Server 200x database to which the project connects. The tool also generates an *association* that corresponds to each foreign-key constraint. Associations eliminate the need to specify joins in LINQ queries that include projections of fields from related tables. The ability to navigate associations is an important feature of LINQ implementations and the Entity Framework.

The advantage of moving from the relational model of tables, rows, and keys to the entity model of entity classes, entity instances, and associations is that entities can — and usually do — represent business objects, such as customers, partners, vendors, employees, products, and services. If your databases are highly normalized, like the AdventureWorks sample database, you'll probably need one or more joins and WHERE clause criteria to represent a customer or employee. Adopting an entity model enables developers to design real-world objects with custom properties, associations, and behaviors suited to the organization's business processes.

*LINQ to SQL is intended for 1:1 mapping of tables:entities, although it does have limited support for entity inheritance of the Table-per-Hierarchy (TPH) type based on discriminator columns. Read-only entities can be based on views or stored procedures. The Entity Framework's Entity Data Model supports joining multiple tables to create a single updatable entity, as well as several types of entity inheritance.*

It's easy to create object data sources from LINQ to SQL classes. Dragging Details View or DataGridView icons from the Data Sources window to a Windows form generates a BindingNavigator for the parent table, and Binding Sources and bound text boxes or DataGridView controls for all tables. The process is almost identical to that for adding DataGridViews or text boxes bound to typed DataSets.

*The LINQforNwind sample projects (\WROX\ADONET\Chapter01\CS \LINQforNwind\LINQforNwindCS.sln and \WROX\ADONET\Chapter01\VB \LINQforNwind\LINQforNwindVB.sln) expect to find a local SQL Server 2005+ instance with the Northwind sample database installed (not attached from a data directory). You'll find instructions for attaching the Northwind.mdf and Northwind\_log.ldf files to SQL Server 2005+ Express in the "Setting Up the Northwind Sample Database" section of this book's introduction.*

### **Mapping Tables to Entities with the LINQ to SQL OR Designer**

Adding a new LINQ to SQL File to your Windows project opens an empty OR Designer surface for the *DatabaseName.dbml* mapping layer file, *Northwind.dbml* for this example. Dragging table icons from the Server Explorer (Database Explorer with Visual Studio 2008 Express versions) to the designer surface adds graphical class tools (widgets) to represent autogenerated entities and connection lines for associations, as shown in Figure 1.5.

## Part I: Getting a Grip on ADO.NET 3.5

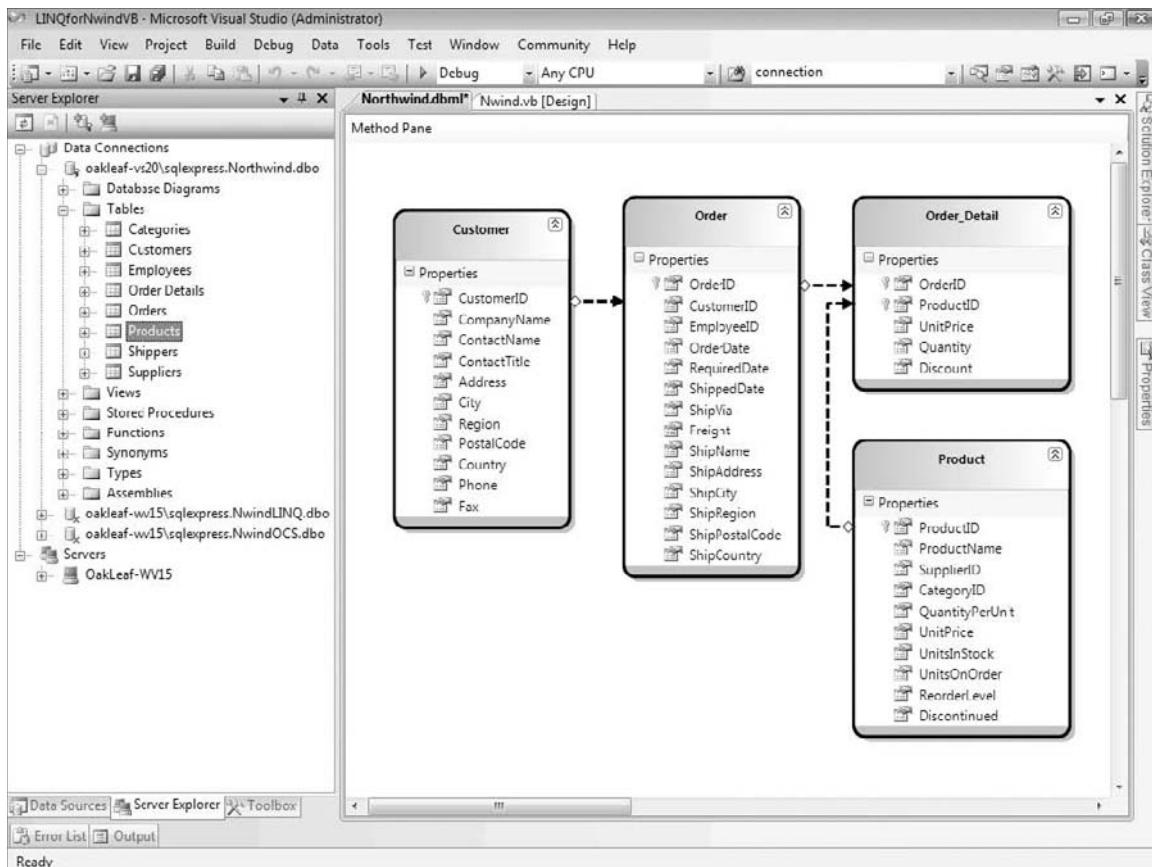


Figure 1-5

The class widgets display entity properties (private members) and enable limited entity customization by setting class and member property values. For example, you can change the entity's Name property value; when you do, the entity set's name changes to the English plural of the new entity name, a process called *pluralization*. The entity also has Delete, Insert, and Update properties to let you use the Configure Behavior dialog to replace the default value (which is Use Runtime for dynamically generated T-SQL) with the name of a user stored procedure for table updates. Chapter 5 describes how to replace dynamic T-SQL generated by the designer with stored procedures for CRUD operations.

*The OR Designer requires building the project or choosing Run Custom Tool to generate the underlying classes before you refer to them in programming code; this requirement doesn't apply to typed DataSets.*

### Examining the Generated Classes

Building the project generates C# or VB partial classes in the `DatabaseName.designer.cs` or `.vb` file for the entities and adds methods for performing CRUD operations on the underlying tables. The top-level `DataContext` class acts as the parent object for the entity sets — `Customers`, `Orders`, `Order_Details` and `Products` for this example. The entity sets' type is `System.Data.Linq.Table<EntityName>`. Therefore, you need a reference to the `System.Data.Linq` namespace. Here's the code to define the read-only `Order_Details` property:

## C# 3.0

```
public global::System.Data.Linq.Table<Order_Detail> Order_Details {  
    get {  
        return this.GetTable<Order_Detail>();  
    }  
}
```

## VB 9.0

```
Public ReadOnly Property Order_Details() As _  
    Global.System.Data.Linq.Table(Of Order_Detail)  
Get  
    Return Me.GetTable(Of Order_Detail)  
End Get  
End Property
```

Autogenerated entity class code uses attributes (emphasized in the following two listings) to map the underlying relational table, columns, and foreign-key constraints to the entity, properties, and associations, as shown in the following abbreviated listings for the `Order_Detail` entity:

## C# 3.0

```
[Table(Name="dbo.[Order Details]")]
public partial class Order_Detail : INotifyPropertyChanging, INotifyPropertyChanged
{
    private static PropertyChangingEventArgs emptyChangingEventArgs =
        new PropertyChangingEventArgs(String.Empty);
    private int OrderID;
    private int ProductID;
    private decimal UnitPrice;
    private short Quantity;
    private float Discount;
    private EntityRef<Order> Order;
    private EntityRef<Product> Product;...
    public Order_Detail()
    {
        this._Order = default(EntityRef<Order>);
        this._Product = default(EntityRef<Product>);
        OnCreated();
    }

    [Column(Storage="_OrderID", DbType="Int NOT NULL", IsPrimaryKey=true)]
    public int OrderID
    {
        get
        {
            return this._OrderID;
        }
        set
```

## Part I: Getting a Grip on ADO.NET 3.5

---

```
{  
    if ((this._OrderID != value))  
    {  
        if (this._Order.HasLoadedOrAssignedValue)  
        {  
            throw new  
                System.Data.Linq.ForeignKeyReferenceAlreadyHasValueException();  
        }  
        this.OnOrderIDChanging(value);  
        this.SendPropertyChanging();  
        this._OrderID = value;  
        this.SendPropertyChanged("OrderID");  
        this.OnOrderIDChanged();  
    }  
}  
}  
...  
[Association(Name="Order_Order_Detail", Storage="_Order",  
    ThisKey="OrderID", OtherKey="OrderID", IsForeignKey=true)]  
public Order Order  
{  
    get  
    {  
        return this._Order.Entity;  
    }  
    set  
    {  
        Order previousValue = this._Order.Entity;  
        if ((previousValue != value)  
            || (this._Order.HasLoadedOrAssignedValue == false)))  
        {  
            this.SendPropertyChanging();  
            if ((previousValue != null))  
            {  
                this._Order.Entity = null;  
                previousValue.Order_Details.Remove(this);  
            }  
            this._Order.Entity = value;  
            if ((value != null))  
            {  
                value.Order_Details.Add(this);  
                this._OrderID = value.OrderID;  
            }  
            else  
            {  
                this._OrderID = default(int);  
            }  
        }  
    }  
}
```

```
        this.SendPropertyChanged("Order");
    }
}
...
}
}
...
...
```

## VB 9.0

```
<Table(Name:="dbo.[Order Details]")> _
Partial Public Class Order_Detail
    Implements System.ComponentModel.INotifyPropertyChanging, _
    System.ComponentModel.INotifyPropertyChanged
    Private Shared emptyChangingEventArgs As PropertyChangingEventArgs = _
        New PropertyChangingEventArgs(String.Empty)
    Private _OrderID As Integer
    Private _ProductID As Integer
    Private _UnitPrice As Decimal
    Private _Quantity As Short
    Private _Discount As Single
    Private _Order As EntityRef(Of [Order])
    Private _Product As EntityRef(Of Product)
    ...
    <Column(Storage:="_OrderID", DbType:="Int NOT NULL", IsPrimaryKey:=true)> _
    Public Property OrderID() As Integer
        Get
            Return Me._OrderID
        End Get
        Set
            If ((Me._OrderID = value) _ 
                = false) Then
                If Me._Order.HasLoadedOrAssignedValue Then
                    Throw New _
                        System.Data.Linq.ForeignKeyReferenceAlreadyHasValueException
                End If
                Me.OnOrderIDChanging(value)
                Me.SendPropertyChanging
                Me._OrderID = value
                Me.SendPropertyChanged("OrderID")
                Me.OnOrderIDChanged
            End If
        End Set
    End Property
    ...
    <Column(Storage:="_UnitPrice", DbType:="Money NOT NULL")> _
    Public Property UnitPrice() As Decimal
        Get
            Return Me._UnitPrice
        End Get
    End Property

```

## Part I: Getting a Grip on ADO.NET 3.5

---

```
Set
If ((Me._UnitPrice = value) =
    = false) Then
    Me.OnUnitPriceChanging(value)
    Me.SendPropertyChanging
    Me._UnitPrice = value
    Me.SendPropertyChanged("UnitPrice")
    Me.OnUnitPriceChanged
End If
End Set
End Property
...
End Class
```

Decorating class and property declarations with attributes from the `System.Data.Linq` namespace results in *attribute-centric mapping*, which is frowned upon by O/RM purists who insist on *persistence ignorance*. *Persistence ignorance* requires the business object layer to be completely unaware of the technique for persisting (saving) objects and to tolerate a change in persistence methodology without code modification. Attributes incorporate table, column, and foreign-key constraints in entity, property, and association declarations. In the event of a change in the underlying relational database schema or vendor, you must recompile the project. Adding attributes results in the O/RM tool creating a *leaky abstraction* in which implementation details “leak through” into the abstraction, in this case the business entity.

*LINQ to SQL also offers a command-line entity-generation tool called `SqlMetal.exe`, which generates the same attribute-based mapping as the O/R Designer. `SqlMetal.exe` offers the option of generating an XML mapping file to eliminate reliance on mapping attributes. However, the persistence ignorance issue probably is moot for LINQ to SQL’s O/RM because SQL Server 200x is the only supported data store. Chapter 5 describes the pros and cons of substituting `SqlMetal` for the O/R Designer.*

Entity classes implement the `INotifyPropertyChanging` and `INotifyPropertyChanged` interfaces to support *change tracking*. Change tracking maintains a record of instance changes that result from altering property (private member) values. The `DataContext` implements change tracking — and maintains state and version information — independently of the entity instances. The `DataContext` includes event handlers for the `OnPropertyChanging` and `OnPropertyChanged` events. If the entity class doesn’t implement these two interfaces, the `DataContext` must create and maintain a copy of each instance, which consumes additional resources.

### **Binding Data-Enabled Controls to Entity Data Sources**

After you add the entity classes and build the project, you can add an Object Data Source for the parent table by choosing Data, Add Data source to open the Data Source Wizard, and selecting Object in the Choose a Data Source Type dialog. In the Select the Object You Wish To Bind To dialog, expanding the LINQforNwind assembly exposes a NorthwindDataContext node (assuming that you named the LINQ to SQL File Northwind.dbml). Expanding the DataContext node, selecting the Customer entity item (see Figure 1-6), and clicking Finish adds the new hierarchical Customer, Order, and Order\_Detail data sources to the Data Sources window.



**Figure 1-6**

The Data Sources window is located on the left in Figure 1-7.

Selecting Details from the Customer list box and dragging the node to the form adds CustomerBindingNavigator, CustomerBindingSource, and text box controls to the form. Text boxes appear in alphabetical order by property name rather than in the original field order (see Figure 1-7 center).

## Part I: Getting a Grip on ADO.NET 3.5

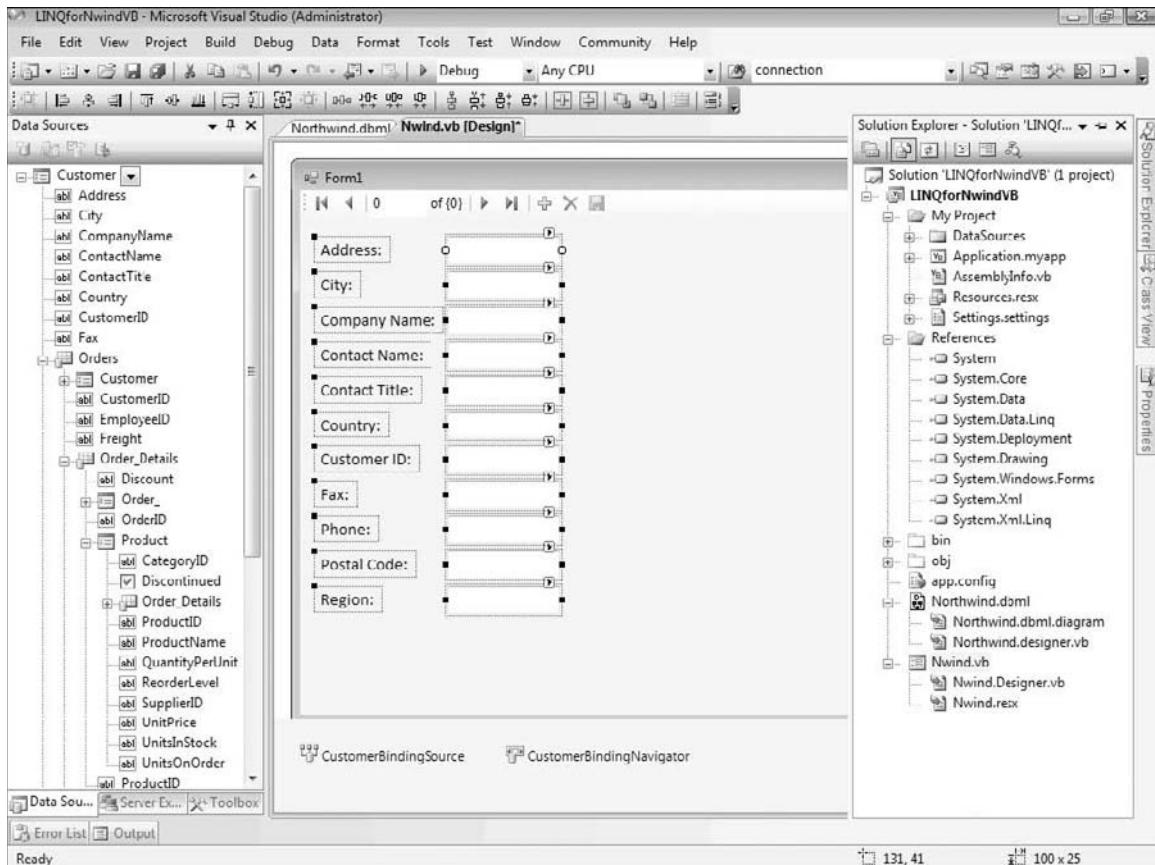


Figure 1-7

Dragging the Orders entity set node from the Data Sources window to the form adds a DataGridView control that displays Order\_ entity values associated with the selected Customer entity. Similarly, dragging the Order\_Details entity set node adds a DataGridView to display line items for the selected Order\_. (The designer adds the underscore suffix to Order because it's a VB reserved word.) Figure 1-8 shows the final version of the sample LINQforNwind project.

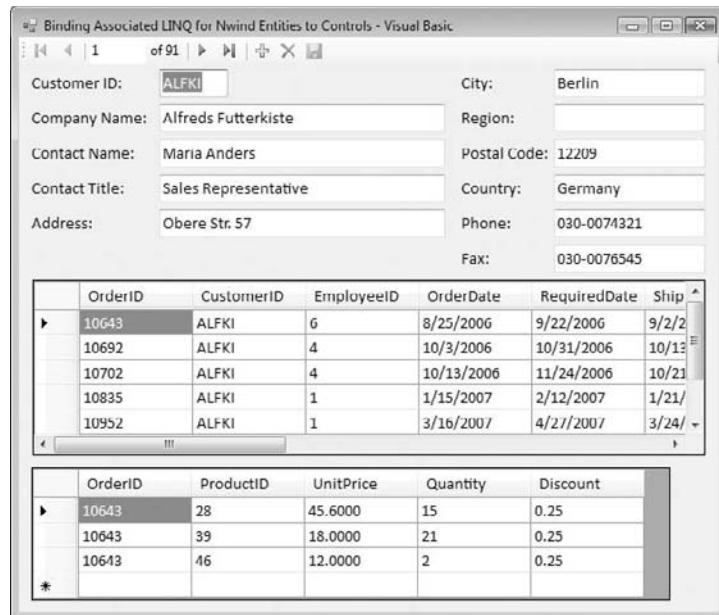


Figure 1-8

The OrderDataGridView control includes a column to contain the entity reference for the `Customer` entity associated with the `Order` entity. (The one side of an association is called an *entity reference* or `EntityRef`, and the many side is called an `EntitySet`.) Similarly, the `Order_DetailDataGridView` has `Order` and `Product` `EntitySet` columns. Removing these columns from the DataGridView controls avoids confusing users with cells that contain type names rather than data.

## Programming the `DataContext`

The `DataContext` object, `NorthwindDataContext` for this example, manages the SQL Server connection. The LINQ to SQL designer adds the connection string to the `app.config` file as `DatabaseNameConnectionString`. Generating the connection string reduces the code required for creating a new `NorthwindDataContext` object and assigning its `Customers` member as the `CustomerBindingSource`'s `DataSource` property to that shown here:

### C# 3.0

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace LINQforNwind
```

## Part I: Getting a Grip on ADO.NET 3.5

---

```
{  
    public partial class Nwind : Form  
    {  
        private NorthwindDataContext dcNwind = new NorthwindDataContext();  
  
        public Nwind()  
        {  
            InitializeComponent();  
        }  
  
        private void Nwind_Load(object sender, EventArgs e)  
        {  
            customerBindingSource.DataSource = dcNwind.Customers;  
        }  
    }  
}
```

You only need to add the two highlighted lines of C# code to that generated by the Windows Form and LINQ to SQL File templates. For the VB version, you must add the `Imports System.Linq` directive yourself, as shown here:

### VB 9.0

```
Imports System.Linq  
  
Public Class Nwind  
    Private dcNwind As New NorthwindDataContext  
  
    Private Sub Nwind_Load(ByVal sender As Object,  
        ByVal e As System.EventArgs) Handles Me.Load  
        CustomerBindingSource.DataSource = dcNwind.Customers  
    End Sub  
End Class
```

## Emulating Joins by Navigating Associations

Navigating entity associations emulates an SQL-92 `INNER JOIN` between the parent table's foreign key and the child table's primary key. LINQ gives you access to entity refs and entity sets with the same dot notation that you use to access members. The following sample query, which uses the same `dcNwind` `DataContext` as the preceding example, illustrates the use of `EntitySets` (`Orders` for a `Customer`, `Order_Details` for an `Order`) and `EntityRefs` (`Product` for an `Order_Detail` in the select list.) A synonym for the `select` list is *projection*.

### C# 3.0

```
namespace LINQforJoins  
{  
    public partial class Joins : Form  
    {  
        private NorthwindDataContext dcNwind = new NorthwindDataContext();  
        public Joins()  
    }  
}
```

## Chapter 1: Taking a New Approach to Data Access in ADO.NET 3.5

```
{  
    InitializeComponent();  
}  
  
private void Joins_Load(object sender, EventArgs e)  
{  
    StringBuilder sbQuery = new StringBuilder();  
  
    var query = from c in dcNwind.Customers  
                from o in c.Orders  
                from d in o.Order_Details  
                where d.ProductID == 1 && c.Country == "USA"  
                orderby o.OrderID descending  
                select new { c.CustomerID, c.CompanyName, o.OrderID, o.OrderDate,  
                           d.Product.ProductName, d.Quantity };  
  
    // Append list header  
    sbQuery.Append("CustID\tOrderID\tOrderDate\t  
                  SkuName\tQuan/r/n");  
    // Append query results  
    foreach (var o in query)  
    {  
        sbQuery.Append(o.CustomerID + "\t" + o.OrderID.ToString() + "\t" +  
                     o.OrderDate.ToString("MM/dd/yyyy") + "\t" + o.ProductName +  
                     "\t" + o.Quantity.ToString() + "/r/n");  
    }  
    // Write to the text box  
    txtResult.Text = sbQuery.ToString();  
    btnExecute.Focus();  
}  
}  
}
```

Here's the VB version of the preceding event handler. Notice the simpler and more natural `From` clause syntax:

### VB 9.0

```
Public Class Joins  
    Private dcNwind As New NorthwindDataContext  
  
    Private Sub Joins_Load(ByVal sender As Object, _  
                           ByVal e As System.EventArgs) Handles Me.Load  
        Dim sbQuery As New StringBuilder  
  
        Dim Query = From c In dcNwind.Customers, o In c.Orders, d In o.Order_Details _  
                    Where d.ProductID = 1 AndAlso c.Country = "USA" _  
                    Order By o.OrderID Descending _  
                    Select c.CustomerID, c.CompanyName, o.OrderID, o.OrderDate, _  
                           d.Product.ProductName, d.Quantity  
  
        ' Append list header  
        sbQuery.Append("CustID" & vbTab & "OrderID" & vbTab & "OrderDate" & vbTab & _  
                      "SkuName" & vbTab & "Quan" & vbCrLf)
```

## Part I: Getting a Grip on ADO.NET 3.5

---

```
' Append query results
For Each o In Query
    sbQuery.Append(o.CustomerID & vbTab & o.OrderID.ToString & vbTab & _
        Format(o.OrderDate, "MM/dd/yyyy") & vbTab & o.ProductName & vbTab & _
        o.Quantity.ToString & vbCrLf)
Next
' Write to the text box
txtResult.Text = sbQuery.ToString
btnExecute.Focus()
End Sub
End Class
```

The preceding queries populate a text box with the following list:

CustID	OrderID	OrderDate	SkuName	Quan
SAVEA	11031	04/17/2007	Chai	45
GREAL	11006	04/21/2007	Chai	46
THECR	11003	04/06/2007	Chai	4
SAVEA	10847	01/22/2007	Chai	80
SAVEA	10700	10/10/2006	Chai	5
LONEP	10317	09/30/2005	Chai	20
RATTC	10294	08/30/2005	Chai	18

To make it easier for LINQ programmers to understand what's going on under the covers, LINQ to SQL's `DataContext` includes a `Log` property that captures the parameterized T-SQL statement sent to the SQL Server database from which you dragged the tables to the designer. Here's the content of the log for the preceding query:

```
SELECT [t0].[CompanyName], [t0].[CustomerID], [t1].[OrderDate], [t1].[OrderID],
    [t3].[ProductName], [t2].[Quantity]
FROM [dbo].[Customers] AS [t0], [dbo].[Orders] AS [t1],
    [dbo].[Order Details] AS [t2], [dbo].[Products] AS [t3]
WHERE ([t3].[ProductID] = [t2].[ProductID]) AND ([t2].[ProductID] = @p0) AND
    ([t0].[Country] = @p1) AND ([t1].[CustomerID] = [t0].[CustomerID]) AND
    ([t2].[OrderID] = [t1].[OrderID])
ORDER BY [t1].[OrderID] DESC
-- @p0: Input Int (Size = 0; Prec = 0; Scale = 0) NOT NULL [1]
-- @p1: Input NVarChar (Size = 3; Prec = 0; Scale = 0) NOT NULL [USA]
SqlProvider\AttributedMetaModel
```

The preceding SQL SELECT query is the payload for SQL Server's `sp_executesql` stored procedure for parameterized dynamic SQL; `@p0` and `@p1` are parameter specifications and values for the query's two parameters.

*There's a pair of sample projects (\WROX\ADONET\Chapter01\CS \LINQforJoins\LINQforJoinsCS.sln and \WROX\ADONET\Chapter01\VB \LINQforJoins\LINQforJoinsVB.sln) to demonstrate the preceding and following examples of LINQ queries with joins.*

To generate the SQL query, LINQ first creates an *expression tree* (also called a *canonical query tree* or CQT) to define the *lambda expressions* that underlie the simplified `from...where...orderby...select` or `From...Where...Order By...Select` syntax. The LINQ to SQL provider, which might be better named *LINQ to T-SQL* or *LINQ to SQL Server* because it's limited to use with Microsoft SQL Server 200x, defers query execution until receiving a request for enumeration, usually by encountering a `foreach` or `For Each` instruction. At that point, a query builder translates the expression tree into the T-SQL statement and invokes the `SqlClient.Command` method to send the query to the database. A `SqlDataReader` object returns the resultset, which the provider uses to populate an entity set.

*Notice that the T-SQL query uses SQL-89 equi-join syntax, which is considerably simpler (and easier to build) than the `Table1 [INNER|OUTER] JOIN Table2 ON Table1.Column1 = Table2.Column2` syntax of SQL-92. Chapter 2 explains lambda expressions and expression trees, which all LINQ queries use. Chapter 5 digs deeper into deferred execution and T-SQL translation.*

## **Loading Child Objects Lazily or Eagerly**

Most O/RM tools have the capability to employ *lazy loading* (also called *deferred loading* or *on-demand loading*) of dependent (child) objects defined by one-to-many associations. Lazy loading defers loading dependent objects, such as `LineItem (Order_Detail)` and `Order` entities for a specific `Customer` object. When the business logic or presentation layer needs the dependant objects, the data layer sends a retrieve command to the data store automatically. Similarly, if all the entity's property values aren't needed, only those that the upper layers request are loaded. If the majority of `Customer` object retrievals don't require associated `Order` and `LineItems` objects or product images in `LineItems` fields, lazy loading improves the application's performance and reduces its resource consumption at the expense of opening and closing a pooled connection for each associated entity. If not, fetching the top-level and all (or a reasonable percentage of) dependent records and compact property values at once is more efficient. This approach is called *eager loading*.

By default, LINQ to SQL lazy-loads all entities and property values by default; by setting the `DataContext.DeferredLoadingEnabled` property value to `True`. You can control the eager-loading mode of entities and properties in code with the `DataLoadOptions` object and the `DataContext.LoadOptions` property value. Chapter 5 explains how to set the `DeferredLoadingEnabled` property value in the O/R Designer and `LoadOptions` property value with code.

## **Creating Explicit Joins with LINQ Join . . . On Expressions**

If you want to relate objects that don't have predefined associations, you can take advantage of LINQ's `join...on/Join...On` syntax. The following T-SQL query uses `SQL-89 JOIN` and `UNION SELECT` statements to generate a list of cities in which Northwind `Customers` and `Suppliers` are located:

```
SELECT c.City AS City, c.CompanyName, 'Customer' AS [Type]
FROM Customers AS c, Suppliers AS s
WHERE c.City = s.City
UNION SELECT s.City AS City, s.CompanyName, 'Supplier'
FROM Customers AS c, Suppliers AS s
WHERE c.City = s.City
ORDER BY City
```

## Part I: Getting a Grip on ADO.NET 3.5

---

Executing the preceding query in SQL Server Management Studio or VS's Server Explorer generates the following 19-row resultset:

City	CompanyName	Type
Berlin	Alfreds Futterkiste	Customer
Berlin	Heli Süßwaren GmbH & Co. KG	Supplier
London	Around the Horn	Customer
London	B's Beverages	Customer
London	Consolidated Holdings	Customer
London	Eastern Connection	Customer
London	Exotic Liquids	Supplier
London	North/South	Customer
London	Seven Seas Imports	Customer
Montréal	Ma Maison	Supplier
Montréal	Mère Paillarde	Customer
Paris	Aux joyeux ecclésiastiques	Supplier
Paris	Paris spécialités	Customer
Paris	Spécialités du monde	Customer
Sao Paulo	Comércio Mineiro	Customer
Sao Paulo	Familia Arquibaldo	Customer
Sao Paulo	Queen Cozinha	Customer
Sao Paulo	Refrescos Americanas LTDA	Supplier
Sao Paulo	Tradição Hipermercados	Customer
(19 row(s) affected)		

You can generate the same result with the following LINQ `join...on` clause, which resembles the SQL-92 and later `INNER JOIN` pattern, and the `Union` operator:

### No longer a problem

#### C# 3.0

```
StringBuilder sbQuery = new StringBuilder();

var query = (
    from c in dcNwind.Customers
    join s in dcNwind.Suppliers on c.City equals s.City
    select new { c.City, c.CompanyName, Type = "Customer" })
    .Union(
    from c in dcNwind.Customers
    join s in dcNwind.Suppliers on c.City equals s.City
    select new { s.City, s.CompanyName, Type = "Supplier" });

foreach (var u in query)
{
    sbQuery.Append(u.City + "\t" + u.CompanyName + "\t" +
        u.Type + "/r/n");
}
// Write to the text box
txtResult.Text = sbQuery.ToString();
```

# Chapter 1: Taking a New Approach to Data Access in ADO.NET 3.5

The VB query code is similar, except for casing and lack of the new constructor for the projection:

## VB 9.0

```
Dim sbQuery As New StringBuilder

Dim Query = ( _
    From c In dcNwind.Customers _
        Join s In dcNwind.Suppliers On c.City Equals s.City _
            Select { c.City, c.CompanyName, .Type = "Customer" }) _
        .Union( _
            From c In dcNwind.Customers _
                Join s In dcNwind.Suppliers On c.City Equals s.City _
                    Select { s.City, s.CompanyName, .Type = "Supplier" }) _

    ' Append list header
    sbQuery.Append("City" & vbTab & "Company Name" & vbTab & "Type" & vbCrLf)
    ' Append query results
    For Each u In Query
        sbQuery.Append(u.City & vbTab & u.CompanyName & vbTab & u.Type & vbCrLf)
    Next
```

Neither version of the union query requires an `orderby/Order By` clause because SQL Server returns the resultset in City, CompanyName order. The two versions substitute `equals/Equals` for the expected `==/=` operator to make the use of the `Object.Equals` comparison explicit.

Here's the T-SQL query generated by the expression tree with parameter values supplied for the literal Type column values:

## T-SQL

```
SELECT [t4].[City], [t4].[CompanyName], @p0 AS [Type]
FROM (
    SELECT [t0].[City], [t0].[CompanyName]
    FROM [dbo].[Customers] AS [t0], [dbo].[Suppliers] AS [t1]
    WHERE [t0].[City] = [t1].[City]
    UNION
    SELECT [t3].[City], [t3].[CompanyName], @p1 AS [Type]
    FROM [dbo].[Customers] AS [t2], [dbo].[Suppliers] AS [t3]
    WHERE [t2].[City] = [t3].[City]
) AS [t4]
-- @p0: Input NVarChar (Size = 8; Prec = 0; Scale = 0) NOT NULL [Customer]
-- @p1: Input NVarChar (Size = 8; Prec = 0; Scale = 0) NOT NULL [Supplier]
SqlProvider\AttributedMetaModel
```

The `GroupJoin` operator (`group...into/Group...Into`) produces a hierarchical result that you process by nested iterators. A `GroupJoin` variation with the `DefaultIfEmpty` operator gives you the equivalent of a left outer join. You can achieve the same result by adding a subquery in the select projection. Chapter 4 offers example of group joins.

### Applying Operators to Emulate SQL Functions, Modifiers, and Operators

The `union()` operator is one of several LINQ standard query operators that correspond to ANSI SQL functions, modifiers, or operators for the `SELECT` list, such as `MIN()`/`Min()`, `MAX()`/`Max()`, `COUNT()`/`Count()`, `SUM()`/`Sum()`, `DISTINCT`/`Distinct()`, `UNION`/`Union()`, `INTERSECT`/`Intersect()`, and `EXCEPT`/`Except()`. T-SQL's `TOP(n)` corresponds to LINQ's `Take(n)`, which you can combine with the `Skip(n)` operator to page the query's output. These operators are methods that operate on the query sequence as a whole, so the generic syntax is:

#### C# 3.0

```
var query = (from element in collection[, from element2 in collection2]
             where condition _
             orderby orderExpression [ascending|descending][, orderExpression2 ...]
             select [alias = ]columnExpression[, [alias2 = ]columnExpression2]
             .Operator([Parameter]) [.Operator2([Parameter2])] [.. . . .]
```

Operators of the appropriate types can be chained, as in `... .Distinct().Count()` or `... .Skip(10).Take(10)`. The operator names use proper capitalization, so the only difference in their VB syntax is the optional empty parentheses:

#### VB 9.0

```
Dim Query = (From Element In Collection[, From Element2 In Collection2] _
              Where Condition _
              Order [By] OrderExpression [Ascending|Descending][, _
              OrderExpression2 ...] _
              Select [Alias = ]ColumnExpression[, _
              [Alias2 = ]ColumnExpression2]) _
              .Operator([Parameter]) [.Operator2([Parameter2])] [.. . . .]
```

Chapters 2, 3, and 4 explain how to use the 50 standard query operators in all 14 categories.

*The two sample projects, \WROX\ADONET\Chapter01\CS\LINQforTesting\LINQforTestingCS.sln and \WROX\ADONET\Chapter01\VB\LINQforTesting\LINQforTestingVB.sln, demonstrate the preceding examples of LINQ queries with aggregate and other operators. These sample projects use Microsoft's ObjectDumper utility (from the 101 Sample Queries demonstration projects) to write the output sequence to the console.*

### Updating Table Data

LINQ to SQL enables `INSERT`, `DELETE`, and `UPDATE` operations on entity instances or sets with `EntitySet.Add(instance)`, `EntitySet.Remove(instance)`, and entity property value changes. The `DataContext`'s entity change tracking system tests for concurrency conflicts when invoking the `SubmitChanges` method to persist the altered entities in the data store. If no conflicts occur, the updates complete within an implicit transaction. Alternatively, updates will run within a open transaction specified by the active `SqlConnection` object.

The following event-handling code in the \WROX\ADONET\Chapter01\CS\LINQforJoins\LINQforJoinsCS.sln project adds, updates, and deletes a customer entry having `BOGUS` as its `CustomerID` value:

## C# 3.0

```
private void btnAddCust_Click(object sender, EventArgs e)
{
    // Add a bogus customer
    dcNwind = new NorthwindDataContext();
    Customer objBogus = new Customer();
    objBogus.CustomerID = "BOGUS";
    objBogus.CompanyName = "Bogus Systems, Inc.";
    objBogus.ContactName = "Joe Bogus";
    objBogus.ContactTitle = "Owner";
    objBogus.Address = "1201 Broadway";
    objBogus.City = "Oakland";
    objBogus.Region = "CA";
    objBogus.PostalCode = "94612";
    objBogus.Country = "USA";
    dcNwind.Customers.InsertOnSubmit(objBogus);

    try
    {
        dcNwind.SubmitChanges();
        ListCustomers();
    }
    catch (Exception exc)
    {
        MessageBox.Show(exc.Message, "LINQ for Joins INSERT Error",
                       MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}

private void btnUpdateCust_Click(object sender, EventArgs e)
{
    try
    {
        // Edit the bogus customer
        dcNwind = new NorthwindDataContext();
        var objBogus = (from c in dcNwind.Customers
                        where c.CustomerID == "BOGUS"
                        select c).First();
        objBogus.CompanyName = "Bogus Enterprises Corp.";
        objBogus.ContactName = "Jack Bogus";
        dcNwind.SubmitChanges();
        ListCustomers();
    }
    catch
    {
        MessageBox.Show("'BOGUS' Customer not present.",
                      "LINQ for Joins UPDATE Error",
                      MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
    }
}

private void btnDeleteCust_Click(object sender, EventArgs e)
```

## Part I: Getting a Grip on ADO.NET 3.5

---

```
{  
    try  
    {  
        // Delete the bogus customer  
        dcNwind = new NorthwindDataContext();  
        var objBogus = (from c in dcNwind.Customers  
                        where c.CustomerID == "BOGUS"  
                        select c).First();  
  
        dcNwind.Customers.Remove(objBogus);  
        dcNwind.SubmitChanges();  
        ListCustomers();  
    }  
    catch  
    {  
        MessageBox.Show("'BOGUS' Customer not present.",  
                    "LINQ for Joins DELETE Error", MessageBoxButtons.OK,  
                    MessageBoxIcon.Exclamation);  
    }  
}  
  
private void ListCustomers()  
{  
    txtResult.Text = "";  
    var Custo...  
    foreach (var cust in Custo...)  
        txtResult.Text += cust.CustomerID + "\t" + cust.CompanyName + "\t" +  
        cust.ContactName + "/r/n";  
}
```

Here's the VB version from the \WROX\ADONET\Chapter01\VB\LINQforJoins\LINQforJoinsVB.sln project:

### VB 9.0

```
Private Sub btnAddCust_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles btnAddCust.Click  
    'Add a bogus customer  
    dcNwind = New NorthwindDataContext  
    Dim objBogus As New Customer  
    With objBogus  
        .CustomerID = "BOGUS"  
        .CompanyName = "Bogus Systems, Inc."  
        .ContactName = "Joe Bogus"  
        .ContactTitle = "Owner"  
        .Address = "1201 Broadway"  
        .City = "Oakland"  
        .Region = "CA"  
        .PostalCode = "94612"
```

## Chapter 1: Taking a New Approach to Data Access in ADO.NET 3.5

```
.Country = "USA"
End With
dcNwind.Customers.Add(objBogus)
Try
    dcNwind.SubmitChanges()
    ListCustomers()
Catch
    MsgBox("'BOGUS' Customer not present.", MsgBoxStyle.Exclamation, _
        "LINQ for Joins INSERT Error")
End Try
End Sub

Private Sub btnUpdateCust_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnUpdateCust.Click
    'Update the bogus customer
    Try
        dcNwind = New NorthwindDataContext
        Dim objBogus = (From c In dcNwind.Customers _
            Where c.CustomerID = "BOGUS" _
            Select c).First()
        With objBogus
            .CompanyName = "Bogus Enterprises Corp."
            .ContactName = "Jack Bogus"
            .ContactTitle = "President"
        End With

        dcNwind.SubmitChanges()
        ListCustomers()
    Catch exc As Exception
        MsgBox(exc.Message, MsgBoxStyle.Exclamation, "LINQ for Joins INSERT Error")
    End Try
End Sub

Private Sub btnDeleteCust_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnDeleteCust.Click
    Try
        dcNwind = New NorthwindDataContext
        Dim objBogus = (From c In dcNwind.Customers _
            Where c.CustomerID = "BOGUS" _
            Select c).First()
        dcNwind.Customers.Remove(objBogus)
        dcNwind.SubmitChanges()
        ListCustomers()

    Catch
        MsgBox("'BOGUS' Customer not present.", MsgBoxStyle.Exclamation, _
            "LINQ for Joins DELETE Error")
    End Try
End Sub

Private Sub ListCustomers()
    txtResult.Text = "
```

## Part I: Getting a Grip on ADO.NET 3.5

---

```
Dim Custs = (From c In dcNwind.Customers _
              Order By c.CustomerID _
              Select c).Take(10)

For Each cust In Custs
    txtResult.Text &= cust.CustomerID & vbTab & _
        cust.CompanyName & vbTab & cust.ContactName & vbCrLf
    Next cust
End Sub
```

The code in the preceding two listings also applies to updating LINQ to Objects instances; just drop the `SubmitChanges` method call.

### **Generating a SQL Server Database with the OR Designer**

Microsoft's current approach to O/RM is *bottom-up*: the underlying database schema drives the design of *business entities* (also called *domain objects*). Business entities, as mentioned earlier, are objects that participate in represent common business activities, such as customers, orders, products, vendors, employees, and candidates (for employment). Most OO software architects and developers prefer to start by designing the domain objects and their associations, and then create the underlying database schema, tables, and relationships from the domain objects specifications. This top-down approach is the foundation of Domain-Driven Design (DDD, also called Domain-Driven Development), a relatively new OO programming methodology.

*Although OO architects and developers prefer to start with domain objects rather than the database schema, probably 10 percent or fewer of all data-intensive development projects today are amenable to DDD (these are often called greenfield projects). Another contributing factor to Microsoft's preference for bottom-up development is the attempt to increase sales of SQL Server licenses. However, LINQ to SQL supports only SQL Server for bottom-up projects, as mentioned earlier, as well as top-down DDD.*

To use the DDD approach, you drag Class widgets from the Object Relational Designer toolbox category to LINQ to SQL's O/R Designer's surface, add properties (fields), and then set their CLR and database datatypes, and other properties. You also can drag association lines that represent foreign-key constraints to the surface, and specify discriminator columns for inheritance. Chapter 5 contains examples of creating SQL Server 200x databases with the `DataContext.CreateDatabase` method.

### **LinqDataSource Control**

ASP.NET's LinqDataSource control is the ASP.NET server control for LINQ to SQL that corresponds to the traditional ObjectDataSource control when data binding data-aware controls on Web forms. Chapter 5's *ASP.NET Databinding with the LinqDataSource Control* section provides a detailed description of the control and its use with ASP.NET 3.5 Web sites and applications.

## **LINQ to DataSets**

The LINQ to DataSets implementation enables querying typed or untyped datasets with LINQ's standard query syntax. Using LINQ to DataSets requires a reference to the `System.Data.DataSetExtensions` namespace to provide the `EnumerableRowCollection<TRow>` class.

# Chapter 1: Taking a New Approach to Data Access in ADO.NET 3.5

---

*The \WROX\ADONET\Chapter01\CS \LINQforDataSets\LINQforDataSetsCS.sln and \WROX\ADONET\Chapter01\VB \LINQforDataSets\LINQforDataSetsVB.sln) to demonstrate the following examples of LINQ for DataSet queries with TableAdapters populated from the Northwind sample database.*

You can sort and filter DataTables as DataViews with string-based queries, such as `DataView.Sort = "OrderDate DESC"` and `DataView.RowFilter = "OrderDate <= '1/1/2008'"`, but DataViews don't support joining DataTables. So one of the primary uses for LINQ to DataSets is delivering complex views over multiple related DataTables. For example, you might need a list of the last 10 orders received from Northwind customers in the USA that include Chai (tea), including the customer name, order number, shipping date, product name (for verification), and number of cartons sold. This result requires joining the DataSet's Customers, Orders, Order Details, and Products DataTables with a T-SQL query such as this:

## T-SQL

```
SELECT TOP(10) c.CustomerID, c.CompanyName, o.OrderID, o.ShippedDate,
p.ProductName, d.Quantity
FROM Customers AS c INNER JOIN Orders AS o ON o.CustomerID = c.CustomerID
    INNER JOIN [Order Details] AS d ON d.OrderID = o.OrderID
    INNER JOIN Products AS p ON p.ProductID = d.ProductID
WHERE p.ProductID = 2 AND c.Country = 'USA'
ORDER BY o.OrderDate DESC
```

However, you can't connect to the Northwind database on Northwind Trader's SQL Server instance, execute the preceding query directly, and return a usable DataTable. But if you've saved a copy of a recent Northwind DataSet with the four required DataTables as an XML file, you can load it into a local DataSet.

## Joining DataSets

LINQ to DataSets lets you join DataTables of strongly typed and untyped DataSets. Strongly typed DataSets enable taking advantage of LINQ's built-in type-checking and implicit type assignment, except for nullable fields of value data types.

## Strongly Typed Data Sets

If the DataSet is strongly typed (`dsNwindT`), here's the LINQ to DataSets query you need to return the same resultset as the preceding T-SQL statement:

### C# 3.0

```
var query = from c in dsNwindT.Customers
            join o in dsNwindT.Orders on c.CustomerID equals o.CustomerID
            join d in dsNwindT.Order_Details on o.OrderID equals d.OrderID
            join p in dsNwindT.Products on d.ProductID equals p.ProductID
            where p.ProductID == 2 && c.Country == "USA"
            orderby o.OrderID descending
            select new { c.CustomerID, c.CompanyName, o.OrderID,
                        ShippedDate = o.Field<DateTime?>("ShippedDate"),
                        p.ProductName, d.Quantity };
```

## Part I: Getting a Grip on ADO.NET 3.5

---

### VB 9.0

```
Dim Query = From c In dsNwindT.Customers _
    Join o In dsNwindT.Orders On c.CustomerID Equals o.CustomerID _
    Join d In dsNwindT.Order_Details On o.OrderID Equals d.OrderID _
    Join p In dsNwindT.Products On d.ProductID Equals p.ProductID _
    Where p.ProductID = 2 AndAlso c.Country = "USA" _
    Order By o.OrderID Descending _
    Select c.CustomerID, c.CompanyName, o.OrderID, _
        ShippedDate = o.Field(Of DateTime?)("ShippedDate"), _
        p.ProductName, d.Quantity
```

The preceding join query, as well as all other variations on the LINQ for DataSets query theme, returns this resultset to the console:

```
{ CustomerID = RATTC, CompanyName = Rattlesnake Canyon Grocery, OrderID = 11077,
    ShippedDate = , ProductName = Chang, Quantity = 24 }
{ CustomerID = SAVEA, CompanyName = Save-a-lot Markets, OrderID = 11030,
    ShippedDate = 4/27/2007 12:00:00 AM, ProductName = Chang, Quantity = 100 }
{ CustomerID = RATTC, CompanyName = Rattlesnake Canyon Grocery, OrderID = 10852,
    ShippedDate = 1/30/2007 12:00:00 AM, ProductName = Chang, Quantity = 15 }
{ CustomerID = SAVEA, CompanyName = Save-a-lot Markets, OrderID = 10722,
    ShippedDate = 11/4/2006 12:00:00 AM, ProductName = Chang, Quantity = 3 }
{ CustomerID = SAVEA, CompanyName = Save-a-lot Markets, OrderID = 10714,
    ShippedDate = 10/27/2006 12:00:00 AM, ProductName = Chang, Quantity = 30 }
{ CustomerID = WHITC, CompanyName = White Clover Markets, OrderID = 10504,
    ShippedDate = 4/18/2006 12:00:00 AM, ProductName = Chang, Quantity = 12 }
{ CustomerID = WHITC, CompanyName = White Clover Markets, OrderID = 10469,
    ShippedDate = 3/14/2006 12:00:00 AM, ProductName = Chang, Quantity = 40 }
{ CustomerID = SAVEA, CompanyName = Save-a-lot Markets, OrderID = 10440,
    ShippedDate = 2/28/2006 12:00:00 AM, ProductName = Chang, Quantity = 45 }
{ CustomerID = SAVEA, CompanyName = Save-a-lot Markets, OrderID = 10393,
    ShippedDate = 1/3/2006 12:00:00 AM, ProductName = Chang, Quantity = 25 }
```

*LINQ's join...on and Join...On syntax use the equals/Equals operator instead of the ==/= operator to emphasize that LINQ joins require object equality rather than scalar value equality.*

### Untyped Data Sets

Untyped DataSets, such as dsNwindU, require designating DataTables by their Name property value, invoking the AsEnumerable() method to return an IEnumerable<T> collection, and typing columns explicitly with the DataTable.Field<T> method:

### C# 3.0

```
var query = from c in dsNwindU.Tables["Customers"].AsEnumerable()
    join o in dsNwindU.Tables["Orders"].AsEnumerable() on
        c.Field<string>("CustomerID") equals o.Field<string>("CustomerID")
    join d in dsNwindU.Tables["Order_Details"].AsEnumerable() on
        o.Field<int>("OrderID") equals d.Field<int>("OrderID")
    join p in dsNwindU.Tables["Products"].AsEnumerable() on
        d.Field<int>("ProductID") equals p.Field<int>("ProductID")
    where p.Field<int>("ProductID") == 2 &&
        c.Field<string>("Country") == "USA"
```

```
orderby o.Field<int>("OrderID") descending
select new { CustomerID = c.Field<string>("CustomerID"),
    CompanyName = c.Field<string>("CompanyName"),
    OrderID = o.Field<int>("OrderID"),
    ShippedDate = o.Field<DateTime?>("ShippedDate"),
    ProductName = p.Field<string>("ProductName"),
    Quantity = d.Field<short>("Quantity") };
```

The VB 9.0 version of the preceding query can take advantage of VB's bang (!, also called the *pling* or *dictionary lookup*) operator to increase the query's terseness. The bang operator only works for collections with default properties. The `DataSet.Tables` collection doesn't have a `default` property, but `DataTable` does; it's `Field`. VB classes with the `Set Option Strict Off` directive don't require strong typing; `Set Option Strict On` and `Set Option Infer On` directives enable inferred typing (`Option Infer On` is the default.) So `DataTable!FieldName` can replace `DataTable.Field(DataType) ("FieldName")` in the following example:

## VB 9.0

```
Dim Query = From c In dsNwindU.Tables("Customers")
    Join o In dsNwindU.Tables("Orders") On _
        c.CustomerID Equals o.CustomerID _
    Join d In dsNwindU.Tables("Order_Details") On _
        o.OrderID Equals d.OrderID
    Join p In dsNwindU.Tables("Products") On _
        d.ProductID Equals p.ProductID _
    Where CInt(p.ProductID) = 2 AndAlso c.Country.ToString = "USA" _
    Order By o.OrderID Descending _
    Select c.CustomerID, c.CompanyName, o.OrderID, _
        o.ShippedDate, p.ProductName, d.Quantity
```

If you want to specify a row version as a criterion of your `where/Where` clause, you must use the overloaded `DataTable.Field<DataType>("FieldName", DataRowVersion)` method in C# or `DataTable.Field(DataType) ("FieldName", DataRowVersion)` in VB queries, as shown in the next section.

## Navigating Entity Set Associations

Invoking a foreign key field's `GetChildRows` method produces a result that's similar to a join, except that the rows returned from a strongly typed `DataSet` are untyped. Navigating one-to-many associations, such as `Customers.CustomerID` to `Orders.CustomerID`, returns an entity set; navigating many-to-one associations created with `GetParentRow`, such as `Order_Details.ProductID` to `Products`. `ProductID` returns an entity reference (`EntityRef`). Following are queries against an untyped `DataSet` that demonstrate `DataRowVersion` and `RowState` filter criteria:

## C# 3.0

```
var query = from c in dsNwindU.Tables["Customers"].AsEnumerable()
    from o in c.GetChildRows("FK_Orders_Customers")
    from d in o.GetChildRows("FK_Order_Details_Orders")
    from p in d.GetParentRows("FK_Order_Details_Products")
    where p.Field<int>("ProductID") == 2
        && c.Field<string>("Country") == "USA"
        && d.Field<short>("Quantity", DataRowVersion.Original) > 1
```

```
&& d.RowState == DataRowState.Unchanged  
orderby o.Field<int>("OrderID") descending  
select new {  
    CustomerID = c.Field<string>("CustomerID"),  
    CompanyName = c.Field<string>("CompanyName"),  
    OrderID = o.Field<int>("OrderID"),  
    ShippedDate = o.Field<DateTime?>("ShippedDate"),  
    ProductName = p.Field<string>("ProductName"),  
    Quantity = d.Field<short>("Quantity") };
```

Notice that the `AsEnumerable()` method isn't required for entities specified by foreign key constraints; the first `AsEnumerable()` invocation applies to child tables also.

VB requires replacing the bang (!) operator with the dot (.) operator when invoking `GetChildRows`:

### VB 9.0

```
Dim Query = From c In dsNwindU.Tables("Customers"), _  
    o In c.GetChildRows("FK_Orders_Customers"), _  
    d In o.GetChildRows("FK_Order_Details_Orders"), _  
    p In d.GetParentRows("FK_Order_Details_Products") _  
    Where CInt(p!ProductID) = 2 _  
        AndAlso c!Country.ToString = "USA" _  
        AndAlso d.Field(Of Short)("Quantity", DataRowVersion.Original) > 1 _  
        AndAlso d.RowState = DataRowState.Unchanged _  
    Order By o!OrderID Descending _  
    Select c!CustomerID, c!CompanyName, o!OrderID, _  
        o!ShippedDate, p!ProductName, d!Quantity
```

The `DataRowVersion` and `DataRowState` criteria are meaningless in this case, because the LINQforDataSets sample projects don't include code to update the `DataTables`. Chapter 5 illustrates code to update `DataSets` and use the new `TableAdapterManager.UpdateAll()` method.

Overloads of the `System.Data.DataSetExtensions.CopyToDataTable()` function let you export `DataRow`s from LINQ to `DataSets` or sequences from other LINQ implementations to new or existing `DataTables` that stand alone or are members of a `DataSet`. You'll find sample code for loading LINQ to Entity sequences into `DataTables` in Chapter 5.

## LINQ to Entities

LINQ to Entities is the last member of the LINQ to ADO.NET trio. LINQ to Entities depends on projects developed with the ADO.NET Entity Data Model template to implement the Entity Framework and its conceptual schema mapped from the database's storage (physical) schema. LINQ to Entities is the optional top member of the Entity Framework stack, operates against the Object Services layer, and translates the query expression to Entity SQL. The initial eSQL version can't handle `INSERT`, or `UPDATE` or `DELETE` queries, so you must use LINQ to Entities to perform these operations.

*This chapter's "The Entity Framework and Entity Data Model" section provides a brief introduction to the Entity Framework and EDM. The book's Part IV, "Introducing the ADO.NET Entity Framework," and Part V, "Implementing the ADO.NET Entity Framework," cover the Entity Framework and EDM thoroughly.*

## Chapter 1: Taking a New Approach to Data Access in ADO.NET 3.5

---

On the surface, the process of executing a LINQ to Entities query and its syntax is identical to that for a LINQ to SQL query. Here are the most significant differences you'll find:

- ❑ `ObjectContext` replaces LINQ to SQL's `DataContext` as the topmost object. If you specified `Northwind` as the Entity Data Model name and accepted the default `NorthwindEntity` as the `ConnectionString` name when you created the EDM from the template, returns the `ObjectContext` with the `Northwind.NorthwindEntity` type.
- ❑ Code generation doesn't singularize table names as entity names; you can change the entity names in a partial class, with the graphical EDM designer, or with a third-party tool such as Huagati Systems' DBML/EDMX Tool utility.
- ❑ `ObjectContext.SaveChanges()` replaces `DataContext.SubmitChanges()`.
- ❑ Creating a new entity replaces `DataContext.TableName.Add(Entity)` method with `ObjectContext.AddObject("EntitySetName", Entity)` and deleting an entity replaces the LINQ to SQL method `DataContext.TableName.Remove(Entity)` with `ObjectContext.DeleteObject(Entity)`. The process for updating an entity remains unchanged.

LINQ to Entities doesn't differ significantly from LINQ to SQL syntax because EF's `ObjectContext` and LINQ to SQL's `DataContext` share similar architectures.

## **LINQ to XML**

One of the initial objectives for LINQ during its early incubation stage as Microsoft Research's Xen and Cω (C-Omega) programming languages for .NET was to unify query syntax for XML documents with that for relational databases and in-memory objects. The seminal "Programming with Circles, Triangles and Rectangles" paper by Eric Meijer of Microsoft's Web Data Team and Wolfram Schulte from Microsoft Research (<http://research.microsoft.com/~emeijer/Papers/XML2003/xml2003.html>), presented at the XML 2003 conference in Philadelphia, proposed a common API with C# extensions for querying objects (circles), XML documents (triangles, representing a hierarchy), and relational databases (rectangles representing tables).

LINQ to XML (formerly *XLinq*) applies the LINQ pattern to VB 9.0 and C# 3.0 queries over well-formed XML InfoSet documents or fragments. LINQ to XML also implements its own in-memory XML Programming API, which represents a substantial improvement to the XML Document Object Model (XML DOM). Unlike other LINQ implementations for specific data domains, which concentrate on extracting data, LINQ to XML lets you compose simple or complex XML documents easily. VB 9.0 has language extensions that make XML composition much simpler than with C# 3.0.

## **LINQ to XML's Objects**

The `System.Xml.Linq` namespace defines bottom-level XML document objects: `XDocument` and `XElement`, both of which inherit from `XContainer`, which in turn inherits from `XNode`. `XElement`, is the LINQ to XML's fundamental object. Unlike the XML DOM, which depends on a topmost  `XmlDocument` object to which you add `XMLElement` objects, use of the `XDocument` object is optional. You can work directly with `XElements` directly by creating them, loading them from a file, or saving them to a writer instance. The most common reason to use an `XDocument` object is to add or include an XML declaration (`XDeclaration`), processing instruction (`XProcessingInstruction`), comment (`XComment`), or any combination of these elements to the start of the document. `XAttribute` objects specify attribute name-value pairs. Figure 1-9 is LINQ to XML's class hierarchy diagram; the hierarchy is determined by the

## Part I: Getting a Grip on ADO.NET 3.5

---

object's inheritance. Although `XDocument` and `XElement` are at the bottom of the hierarchy, these two objects are fundamental to LINQ to XML.

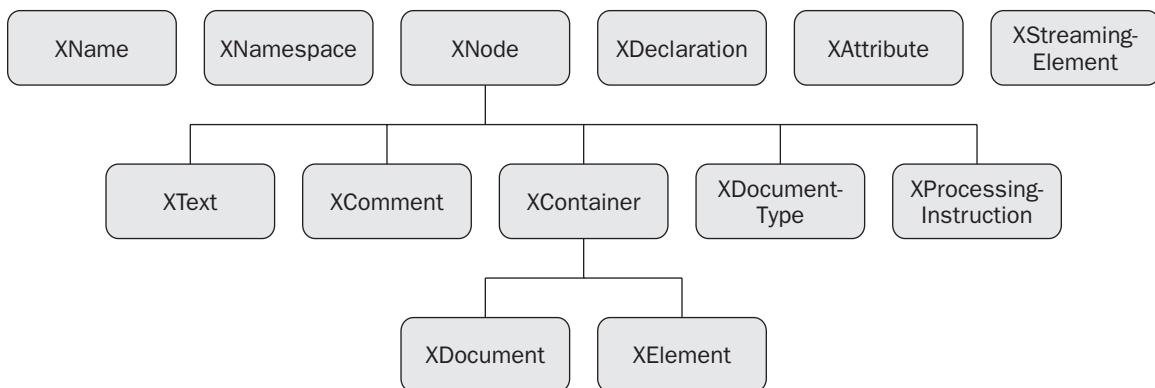


Figure 1-9

Using LINQ to XML features requires a reference to the `System.Xml.Linq` namespace, which is present by default in VB 9.0 and C# 3.0 projects.

### Querying XML Documents

The most common use of LINQ to XML is for composing or transforming XML documents but this chapter's primary purpose is to illustrate the LINQ query syntax that's common to all data domains that Microsoft LINQ implementations support. You use the `XDocument.Load(uri[, LoadOptions])` or `XElement.Load(uri[, LoadOptions])` to import an existing XML document from a file, URL, `TextReader` or `XmlReader`.

*LINQforXMLQuery1CS.sln, LINQforXMLQuery2CS.sln, LINQforXMLQuery1VB.sln, and LINQforXMLQuery2VB.sln are the sample projects for this section located in the \WROX\ADONET\Chapter01\CS and VB folders, respectively. The XML source documents are in \WROX\ADONET\Chapter7\Data. Chapter 7, "Manipulating XML Documents with LINQ to XML," delivers detailed coverage of the LINQ to XML implementation.*

Here's the code to open an Orders document and generate an XML fragment that contains all Order elements for U.S. customers in descending OrderID order:

#### C# 3.0

```
namespace LINQforXMLQuery1
{
    class LINQforXML
    {
        static void Main(string[] args)
        {
            // Return an XML fragment containing US orders in descending date order
            XDocument xdOrders =
```

Chapter 1: Taking a New Approach to Data Access in ADO.NET 3.5

```
XDocument.Load(@"\Wrox\ADONET\Chapter07\Data\Nwind\Orders.xml",
LoadOptions.PreserveWhitespace);

var query = from o in xdOrders.Descendants("Order")
           where o.Element("ShipCountry").Value == "USA"
           orderby o.Element("OrderID").Value descending
           select o;

// Write out the fragment
foreach (var o in query)
    Console.WriteLine(o);
Console.ReadLine();
}

}
```

The `xdOrders.Descendants("ElementName")` statement returns an `IEnumerable< XElement >` type of the child elements of an `XDocument` or `XElement` type, `<Order>` for this example. `o.Element("ElementName")` returns an `XElement`; the `Value` property returns the element's text as a string. (The `Value` property returns the text of the first item of an `IEnumerable< XElement >` or `IEnumerable< XAttribute >` object. The source document is late-bound, so there's no IntelliSense for element names. LINQ to XSD, which is the subject of a later section, enables strong typing and IntelliSense from an XML schema for the document.

VB 9.0 has shortcut syntax, which Microsoft calls *XML axis properties*, for the Descendants and Element methods as shown in the second query expression of this example:

VB 9.0

```
Module LINQforXMLQuery1
Sub Main()
    ' Return an XML fragment containing US orders in descending date order
    Dim xdOrders As XDocument = _
        XDocument.Load("\WROX\ADONET\Chapter07\Data\Nwind\Orders.xml", _
        LoadOptions.PreserveWhitespace)

    ' Conventional XML query expression syntax
    Dim Conven = From o In xdOrders.Descendants("Order") _
                  Where o.Element("ShipCountry").Value = "USA" _
                  Order By o.Element("OrderID").Value Descending _
                  Select o

    ' VB axis properties XML query expression syntax
    Dim Shortcut = From o In xdOrders...<Order> _
                   Where o.<ShipCountry>.Value = "USA" _
                   Order By o.<OrderID>.Value Descending _
                   Select o

    ' Write out the fragment
    For Each o In Shortcut
        Console.WriteLine(o)
    Next
    Console.ReadLine()
End Sub
End Module
```

## Part I: Getting a Grip on ADO.NET 3.5

---

The `xdOrders...<Order>` descendants axis property is similar to the conventional `xdOrders.Descendants("Order")` syntax. It finds the `<Order>` element no matter how deeply it's nested. The `o.<OrderID>.Value` child axis property is equivalent to `o.Element("ShipCountry").Value.o.@OrderID`

*Adding an XML schema for the source document to your project enables IntelliSense for VB's axis property syntax. Chapter 7 shows you how to infer a schema with VS 2008's XML editor and add it to your VB project.*

You can join XML objects on child elements having common values with code that follows the LINQ to SQL pattern, such as the following:

### C# 3.0

```
namespace LINQforXMLQuery2
{
    class LINQforXML
    {
        static void Main(string[] args)
        {
            // Load and join three XML files and project primary and foreign keys
            XDocument xdCustomers =
                XDocument.Load(@"\WROX\ADONET\Chapter07\Data\Nwind\Customers.xml",
                LoadOptions.PreserveWhitespace);
            XDocument xdOrders =
                XDocument.Load(@"\WROX\ADONET\Chapter07\Data\Nwind\Orders.xml",
                LoadOptions.PreserveWhitespace);
            XDocument xdDetails =
                XDocument.Load(@"\WROX\ADONET\Chapter07\Data\Nwind\OrderDetails.xml",
                LoadOptions.PreserveWhitespace);

            var query = from c in xdCustomers.Descendants("Customer")
                        join o in xdOrders.Descendants("Order") on
                            c.Element("CustomerID").Value equals
                            o.Element("CustomerID").Value
                        join d in xdDetails.Descendants("OrderDetail") on
                            o.Element("OrderID").Value equals
                            d.Element("OrderID").Value
                        where c.Element("Country").Value == "USA" &&
                            (int) d.Element("ProductID") < 30
                        orderby o.Element("OrderID").Value descending
                        select new { CustID = c.Element("CustomerID").Value,
                                    OrderID = o.Element("OrderID").Value,
                                    ProductID = d.Element("ProductID").Value
                                };

            // Write out the values
            foreach (var o in query)
                Console.WriteLine(o);
            Console.ReadLine();
        }
    }
}
```

## VB 9.0

```
Module LINQforXMLQuery2
    Sub Main()
        ' Load and join 3 XML files on project primary and foreign keys
        Dim xdCustomers As XDocument = _
            XDocument.Load("\WROX\ADONET\Chapter07\Data\Nwind\Customers.xml", _
            LoadOptions.PreserveWhitespace)
        Dim xdOrders As XDocument = _
            XDocument.Load("\WROX\ADONET\Chapter07\Data\Nwind\Orders.xml", _
            LoadOptions.PreserveWhitespace)
        Dim xdDetails As XDocument = _
            XDocument.Load("\WROX\ADONET\Chapter07\Data\Nwind\OrderDetails.xml", _
            LoadOptions.PreserveWhitespace)

        ' Axis properties XML query expression syntax
        Dim query = From c In xdCustomers.Descendants("Customer") _
                    Join o In xdOrders.Descendants("Order") _
                        On c.Element("CustomerID").Value Equals _
                        o.Element("CustomerID").Value() _
                    Join d In xdDetails.Descendants("OrderDetail") _
                        On o.Element("OrderID").Value Equals _
                        d.Element("OrderID").Value() _
                    Where d.<OrderID>.Value = o.<OrderID>.Value AndAlso _
                        o.<ShipCountry>.Value = "USA" AndAlso _
                        CInt(d.<ProductID>.Value) < 30 _
                    Order By o.<OrderID>.Value Descending _
                    Select New With {.CustomerID = o.<CustomerID>.Value, _
                        .OrderID = o.<OrderID>.Value, _
                        .ProductID = d.<ProductID>.Value}

        ' Write out the values
        For Each o In query
            Console.WriteLine(o)
        Next
        Console.ReadLine()
    End Sub
End Module
```

Here's a sampling of the first and last rows generated by the preceding two sample projects, which only generate values for orders that have OrderDetail.ProductID values less than 30:

```
{ CustID = THECR, OrderID = 10624, ProductID = 29 }
{ CustID = SAVEA, OrderID = 10612, ProductID = 10 }
{ CustID = SAVEA, OrderID = 10607, ProductID = 7 }
{ CustID = SAVEA, OrderID = 10607, ProductID = 17 }
{ CustID = SAVEA, OrderID = 10603, ProductID = 22 }
{ CustID = RATTC, OrderID = 10598, ProductID = 27 }
...
{ CustID = RATTC, OrderID = 10294, ProductID = 1 }
{ CustID = RATTC, OrderID = 10294, ProductID = 17 }
{ CustID = RATTC, OrderID = 10272, ProductID = 20 }
{ CustID = RATTC, OrderID = 10262, ProductID = 5 }
{ CustID = RATTC, OrderID = 10262, ProductID = 7 }
```

## Part I: Getting a Grip on ADO.NET 3.5

---

You also can join XML objects with entities you generate with LINQ to Objects, LINQ to SQL or LINQ to Entities — or vice versa.

### Transforming or Creating XML Documents

As mentioned earlier, developers use LINQ to XML primarily for transforming or composing XML documents. Generating XML content often involves embedding LINQ to XML, LINQ to SQL, or LINQ to Entities queries to supply element or attribute values. Microsoft calls the standard technique for adding elements *functional construction*. Functional construction combines starting with an optional XDocument object to support an XML declaration, comment, or processing directive, a root XElement element, and adding XAttributes or child XElements with indentation that corresponds approximately to that of the document when opened in Internet Explorer (IE) with its XML stylesheet. When execution reaches a code block that returns an IEnumerable< XElement> or, less commonly, an IEnumerable< XAttribute> type, the compiler designates the block as a source of elements or attributes to add to the document under construction.

The following sample project builds on LINQforXMLQuery1CS.sln by replacing the missing <Orders> root element for the U.S. <Order> fragments subset with an XML declaration, comment, and new root element having an xmlns:xsi namespace prefix designation, as well as xsi: noNamespaceSchemaLocation and generated dateTime (ISO 8601 format) attribute/value pairs. The following example saves the transformed XML document as a file and opens it in IE with a Process .Start(fileName.xml) instruction. The Process object is a member of the System.Diagnostics namespace.

#### C# 3.0

```
namespace LINQforXMLQuery3
{
    class LINQforXML
    {
        static void Main(string[] args)
        {
            string FilePath = @"WROX\ADONET\Chapter07\Nwind\Data\";
            XDocument xdOrders = XDocument.Load(FilePath + "Orders.xml",
                LoadOptions.PreserveWhitespace);
            // Define the xmlns:xsi namespace and prefix
            XNamespace xsi = "http://www.w3.org/2001/XMLSchema-instance";
            // Wrap the Order fragments with a root element
            XDocument Orders =
                new XDocument(
                    // Add an XML declaration
                    new XDeclaration("1.0", "utf-8", "yes"),
                    // Add a descriptive comment
                    new XComment("Created with LINQ to XML"),
                    // Start the root element
                    new XElement("Orders",
                        // Add the xmlns:xsi namespace and prefix
                        new XAttribute(XNamespace.Xmlns + "xsi", xsi),
                        // Add the noNamespaceSchemaLocation with xsi prefix
                        new XAttribute(xsi + "noNamespaceSchemaLocation",
                            FilePath + "Orders.xsd"),
                        xdOrders));
            Orders.Save(FilePath + "OrdersTransformed.xml");
        }
    }
}
```

```
// Add the created attribute with ISO8601 date
new XAttribute("generated",
    DateTime.Now.ToUniversalTime().ToString("s")),
// Add the <Order> elements for U.S. customers
from o in xdOrders.Descendants("Order")
where o.Element("ShipCountry").Value == "USA"
orderby o.Element("OrderID").Value descending
select o);

Orders.Save("Orders.xml");
Process.Start("Orders.xml"); // Display the document in IE

}
}

}
```

Here's the equivalent VB 9.0 functional construction code that uses axis properties:

## VB 9.0

```
Module LINQforXMLQuery3
Sub Main()
    Dim FilePath As String = "\WROX\ADONET\Chapter07\Nwind\Data\
    Dim xdOrders As XDocument = XDocument.Load(FilePath + "Orders.xml", _
        LoadOptions.PreserveWhitespace)
    ' Add an XML declaration, descriptive comment, root element,
    ' xmlns:xsi namespace and prefix, xsi:noNamespaceSchemaLocation
    ' attribute, created attribute, and <Order> elements from the U.S.
    Dim xsi As XNamespace = "http://www.w3.org/2001/XMLSchema-instance"
    Dim Orders = New XDocument( _
        New XDeclaration("1.0", "utf-8", "yes"), _
        New XComment("Created with LINQ to XML"), _
        New XElement("Orders", _
            New XAttribute(XNamespace.Xmlns + "xsi", xsi), _
            New XAttribute(xsi + "noNamespaceSchemaLocation", _
                FilePath + "Orders.xsd"), _
            New XAttribute("generated", _
                DateTime.Now.ToUniversalTime.ToString("s")), _
            From o In xdOrders...<Order> _
            Where o.<ShipCountry>.Value = "USA" _
            Order By o.<OrderID>.Value Descending _
            Select o))

    Orders.Save("Orders.xml")
    Process.Start("Orders.xml") 'Display the document in IE
End Sub
End Module
```

Figure 1-10 shows IE displaying the Orders.xml document created by the preceding two projects. The root element is identical to that of the source Orders.xml document except for the xsi: noNamespaceSchemaLocation attribute's value and the generated attribute's date and time.

## Part I: Getting a Grip on ADO.NET 3.5



Figure 1-10

VB 9.0 offers an alternative syntax to generate XML documents that substitutes XML literals for functional construction code. The advantage of this approach is the ease with which you can generate documents from an existing XML InfoSet model: Just copy and paste it into your code. The VB compiler detects the XML block, determines if the XML is well-formed, and parses namespaces and prefixes, if present. Expressions enclosed between `<%= ... %>` pairs supply XML fragment blocks, as in the following example, or element/attribute values.

```
Module LINQforXMLQuery4
    Sub Main()
        Dim FilePath As String = "\WROX\ADONET\Chapter07\Data\
        Dim xdOrders As XDocument = XDocument.Load(FilePath + "Orders.xml", _
            LoadOptions.PreserveWhitespace)
        ' Use VB 9.0's literal XML syntax with an embedded query to
        ' generate the Orders document
        Dim Orders As XDocument = _
            <?xml version="1.0" encoding="utf-8" standalone="yes"?>
            <!-- Created with LINQ to XML -->
            <Orders xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" _
                xsi:noNamespaceSchemaLocation=<%= FilePath + "Orders.xsd" %>
                generated=<%= DateTime.Now.ToString("s") %>>
                <%= From o In xdOrders.Descendants("Order") _>
                    Where o.Element("ShipCountry").Value = "USA" _>
```

```
Order By o.Element("OrderID").Value Descending _  
Select o %>  
</Orders>  
  
Orders.Save("Orders.xml")  
Process.Start("Orders.xml") 'Open in IE  
End Sub  
End Module
```

The preceding code produces the same document as the two LINQforXMLQuery3 examples with considerably less development effort. Even C# aficionados should consider using VB 9.0 when making extensive use of LINQ to XML.

### LINQ to XSD

LINQ to XSD is one of several Microsoft data programmability *incubator projects*. Incubator projects usually result from an idea proposed by a programmer, manager, or small research or development team that shows promise of maturing into a new product or an add-in to an existing product. Microsoft doesn't guarantee that incubator projects will mature to supported software and might be discontinued without notice. As noted early in the chapter, Cu (an ancestor of LINQ) began as a Microsoft Research incubator project.

One of the objectives of LINQ technology is to eliminate the use of string literals and casting from `Object` in data programming. String literals result in late binding and are said to make data-intensive applications *brittle*, which means they're subject to run-time failures due to the lack of compile-time checks. The connection string probably is here to stay, but LINQ to SQL, for example, lets you do away with SQL statements and column name literals, strongly types queries, and enables IntelliSense and statement completion.

LINQ to XML query and functional construction code in C# and by default in VB makes extensive use of literal strings and casting from `Object` to `String` and value types. LINQ to XSD is a helper technology, which generates classes from XML schemas to model typed views on untyped XML trees. Instances of LINQ to XSD classes wrap the `XElement` class; LINQ to XSD classes have a common `XTypedElement` base class.

*Search the Web for "LINQ to XSD Preview" to find the download page for the current LINQ to XSD version. (Alpha 0.2 was the version available when this book was written.) If you don't install the LINQ to XSD bits, you won't be able to open the LINQforXMLQuery1CS.sln or LINQforXMLQuery1VB.sln sample projects.*

Running the LINQ to XSD.msi installer adds a LINQ to XSD Preview item to the Programs menu with documentation and sample projects choices. It also adds a LINQ to XSD Preview node to the Visual C# and Visual Basic Project Types lists. Selecting LINQ to XSD lets you choose a new LINQ to XSD Console Application, Windows Application, or Library. The new project includes a reference to `Microsoft.Xml.Schema.Linq.dll` as well as to the standard .NET Fx 3.5 `System.Xml.Linq` and `System.Xml` references.

Generating the LINQ to XSD classes requires adding the XML schema file (`Orders.xsd` for this example) to your project and changing its Build Action property value from None to `LinqToXmlSchema` so it's treated as source code. Building the project then generates a `LinqToXsdSource.cs` file in the project's ... \obj\debug folder to define the CLR classes that add strong typing to the XML object.

## Part I: Getting a Grip on ADO.NET 3.5

---

Following is the C# code for the strongly typed version of LINQforXSDQuery1CS.sln project:

### C# 3.0

```
namespace LINQforXSDQuery1CS
{
    class LINQtoXSD
    {
        static void Main(string[] args)
        {
            // Return an XML fragment containing US orders in descending order
            var orders = Orders.Load(@"..\..\Orders.xml");

            var query = from o in orders.Order
                        where o.ShipCountry == "USA"
                        orderby o.OrderID descending
                        select o;

            // Write out the fragment
            foreach (var o in query)
                Console.WriteLine(o);
            Console.ReadLine();
        }
    }
}
```

*LINQ to XSD Alpha 0.2 doesn't support VB 9.0, which is a shame, because VB 9.0 is the preferred language for working with XML documents.*

It's clear from the preceding example that LINQ to XSD lifts LINQ to XML to the same level as LINQ to SQL and LINQ to Entities by basing object member typing on similar metadata sources. Chapter 7 provides detailed instructions for adding strong typing to LINQ to XML's objects with the LINQ to XSD version that was current when this book was written.

## The ADO.NET Entity Framework and Entity Data Model

Microsoft's ADO.NET team began designing the Entity Framework (EF) and Entity Data Model (EDM) near the date in 2003 when the C# 3.0 design group decided to develop a LINQ to SQL extension as a "stand-in" for the abandoned ObjectSpaces O/RM tool project. The two projects progressed in parallel, apparently with little or no communication between the two groups. In early 2006, the ADO.NET team gained ownership of the LINQ to SQL and LINQ to DataSet implementations and added LINQ to Entities as an optional method for executing queries over the EDM. The ADO.NET group presented several sessions at Tech\*Ed 2006 and published a series of whitepapers about EF and EDM in June 2006. The "Next-Generation Data Access: Making the Conceptual Level Real" technical article by José Blakeley, David Campbell, Jim Gray, S. Muralidhar, and Anil Nori, proposed to extend the EDM from basic CRUD operations to reporting and analysis services (business information) and data replication.

## Chapter 1: Taking a New Approach to Data Access in ADO.NET 3.5

---

You can read the preceding technical paper at [http://msdn2.microsoft.com/en-us/architecture/aa730866\(vs.80\).aspx](http://msdn2.microsoft.com/en-us/architecture/aa730866(vs.80).aspx) and its companions, "The ADO.NET Entity Framework Overview" at [http://msdn2.microsoft.com/en-us/architecture/aa697427\(vs.80\).aspx](http://msdn2.microsoft.com/en-us/architecture/aa697427(vs.80).aspx) and "ADO.NET Tech Preview: Entity Data Model" at [http://msdn2.microsoft.com/en-us/architecture/aa697428\(vs.80\).aspx](http://msdn2.microsoft.com/en-us/architecture/aa697428(vs.80).aspx).

EF is a much more than an O/RM tool. EF provides a complete set of data services including:

- Query services
- ClientView services
- Persistence services
- Object services

EF is the first concrete implementation of the EDM and enables developers to abstract the *physical storage schema* of relational databases to a *conceptual schema* (also called a *conceptual layer* or conceptual model) that conforms to the entity-relationship (E-R) data model. Dr. Peter Chen proposed the E-R data model in a 1976 paper, "The Entity-Relationship Model — Toward a Unified View of Data," which you can read at <http://bit.csc.lsu.edu/~chen/pdf/erd.pdf>. E-R became a mainstay of data modeling for relational databases and the foundation for many computer-aided software engineering (CASE) tools.

The E-R model is independent of the underlying relational database management system (RDBMs) or data retrieval and update language, syntax or dialect, but an SQL flavor is assumed. Microsoft says that EF "is an execution runtime for the for the E-R model" that's RDBMS-agnostic. As you would expect, SQL Server 200x and SQL Server Compact Edition (SSCE) have `SqlClient`-derived `EntityClient` data providers for EF v1.0. When this book was written, IBM DB2, IS, and U2, as well as Oracle, MySQL, and SQLite supported EF with custom `EntityClient` data providers for .NET. The `EntityClient` object provides `EntityConnection`, `EntityCommand`, `EntityParameter` and `EntityDataReader` objects. `EntityClient` has its own query language, *Entity SQL* (eSQL), which is related to SQL but has entity-specific extensions. eSQL v1 has no data management language (DML) constructs for `INSERT`, `UPDATE` or `DELETE` operations. Microsoft promises to implement entity-compatible DML in a future version.

An EDM `EntityType` specifies an object in the application domain that is represented by data. An `EntityObject` is an instance of an `EntityType`, has one or more `Properties`, and is uniquely identified by an `EntityKey`. `Properties` can be of `SimpleType`, `ComplexType` or `RowType`. An `EntitySet` contains multiple `EntityObject` instances, an `EntityContainer` holds multiple `EntitySets`, and corresponds to the database (such as Northwind) at the physical data store layer and schema (such as dbo) at the conceptual level. A `RelationshipType` defines the relationship of two or more `EntityTypes` as an `Association`, which is a peer-to-peer relationship or `Containment`, that specifies a parent-child relationship. The top-level EDM object is a `Schema`, which represents the entire database.

## Part I: Getting a Grip on ADO.NET 3.5

---

Following are some of the most important benefits of implementing EDM in data-intensive .NET projects:

- ❑ Generates a data access layer that isolates the data domain from the application domain
- ❑ Handles relational database vendor or schema changes without the need to alter C# or VB source code and recompile the project
- ❑ Enables the business logic layer to work with real-world business entities, such as customers, orders, employees, and products, rather than rows and tables of relational databases
- ❑ Supports complex properties, such as PhysicalAddress, which might include Street1, Street2, City, State, ZipCode, Country and Coordinates, and PostalAddress, which might include only PostOfficeBox, City, State, PostalCode and Country.
- ❑ Models object-oriented concepts such as inheritance and hierarchical (nested) or polymorphic resultsets, which don't fit the relational model
- ❑ Exposes relationship navigation properties to eliminate the need for complex joins between related tables
- ❑ Provides an advanced, graphical EDM Designer to simplify inheritance and hierarchy models and enable incremental changes to mapping files
- ❑ Maps the conceptual layer's entity model to an object model in the Object Services layer with an `ObjectContext` for identity management and change tracking
- ❑ Enables simple, strongly typed queries with Object Services' LINQ to Entities implementation

It's important to bear in mind that EF isn't a traditional O/RM tool and using the Object Services layer and LINQ to Entities is optional. Mapping to objects occurs at the conceptual layer, so the term *object/entity mapping tool* might represent a more accurate description. Where performance is critical, such as with reporting and analysis services, you can opt to write eSQL queries and return conventional, untyped DataReaders based on the conceptual layer's schema.

*As mentioned in the earlier "LINQ to Entities" section, this chapter provides only a brief introduction to EF and EDM. The book's Part IV, "Introducing the ADO.NET Entity Framework," and Part V, "Implementing the ADO.NET Entity Framework," cover the EF and EDM thoroughly.*

## **Mapping from the Physical to the Conceptual Schema**

Like its predecessor, ObjectSpaces, EDM uses three XML files to map from the physical to the conceptual schema. Figure 1-11 is a simplified EF block diagram that shows the three schema layers that perform the mapping, the optional Object Services layer and LINQ to Entities component, and the EntityClient object that translates eSQL queries to dynamic SQL queries or stored procedure calls in the RDBMS's dialect.

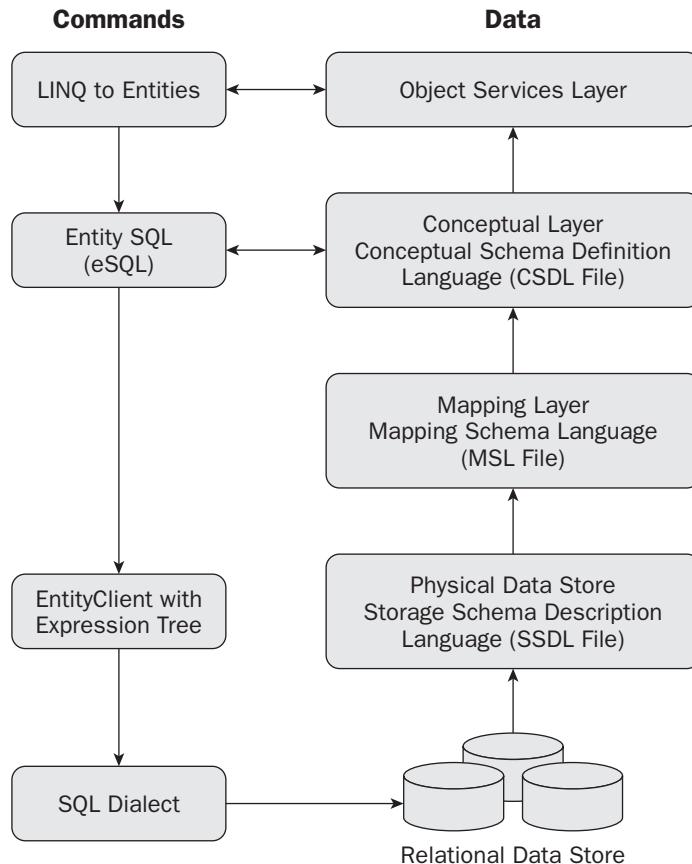


Figure 1-11

By default, the EDM Wizard generates an EntityClient connection string named *ModelNameEntities*, adds it to the App.config file and creates the *ModelName.edmx* file to generate the following three XML mapping files:

- ❑ *ModelName.ssdl* defines the physical data store layer in Store Schema Definition Language (SSDL). The top-level `<Schema>` element's `Namespace` attribute value is `DatabaseName` (Northwind for this example), the `<EntityContainer>` element's `Name` attribute value is the Schema name (`dbo` for this example) and the `<Property>` element's `Type` attribute values are datatypes of the database.
- ❑ *ModelName.csdl* defines the conceptual layer in Conceptual Schema Definition Language (CSDL). The top-level `<Schema>` element's `Namespace` attribute value becomes `ModelNameModel` (NorthwindModel for this example), the `<EntityContainer>` element's `Name` attribute value becomes `ModelNameEntities` (NorthwindEntities for this example) and the `<Property>` element's `Type` attribute values correspond to CLR data types.
- ❑ *ModelName.msl* specifies the mapping layer between the Store Schema and Conceptual Schema in Mapping Schema Language (MSL). The mapping file enables entities, properties, and association sets to have different names than their corresponding tables, fields, and foreign-key constraints.

## Part I: Getting a Grip on ADO.NET 3.5

The EDM Wizard generates and stores a ModelName.edmx file and stores it in the main project folder for use by the EDM Designer. ModelName.edmx is an XML file with groups for the SSDL, MSL and CSDL content. When you build the project, MSBuild generates the individual ModelName.ssdl, ModelName.msl and ModelName.csdl files and stores them as project resources.

Developers or DBAs modify the .edmx file when the underlying relational schema changes by right-clicking the EDM Designer surface and choosing the Update Database from Schema option. Minor schema changes don't require source code modifications.

You can view the Northwind.edmx mapping file for the Northwind sample database in the \WROX\ADONET\Chapter01\CS\LINQforEntities\LINQforEntitiesCS.sln and \WROX\ADONET\Chapter01\VB\LINQforEntities\LINQforEntitiesVB.sln sample projects by right-clicking it in Solution Explorer, choosing Open With, and then selecting XML Editor. The CSDL content section provides the data for autogenerated the Northwind.cs/.vb class file.

## **Creating the Default EDM with the Entity Data Model Wizard**

Adding a new item to a C# or VB project based on the ADO.NET Entity Data Model template, replacing the default Model1.edmx model name with Northwind for this example, and clicking Add opens the Entity Data Model Wizard's Choose Model Contents dialog. Accepting the default Generate from Database icon and clicking Next opens the Choose Your Database Connection dialog with the EDM version of the last connection string you created in VS as the default (named NorthwindEntities). Clicking Next opens the Choose Your Database Objects dialog with a checked list to select the database's tables (see Figure 1-12). Unless you have a reason not to do so, it's the most common practice to accept the default selection of all tables. Click Finish to generate Northwind.edmx XML file, and press F5 to build and run the project and create a Northwind.cs or .vb class file to populate the NorthwindModel namespace.

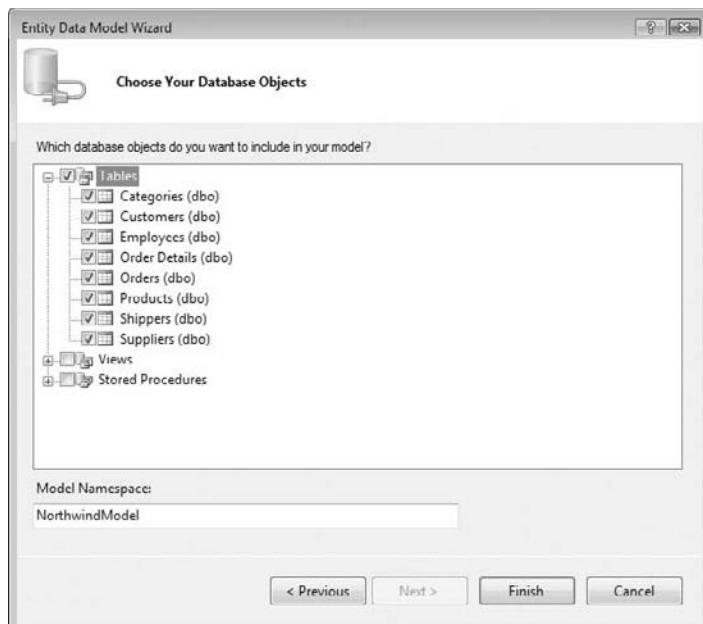


Figure 1-12

The default EDM's namespace defines a partial public class (NorthwindEntities), which inherits Global.System.Data.Objects.ObjectContext and has a read-only property for each EntitySet (Categories for example). The getter invokes the `CreateQuery<EntitySet>("eSQL")` method, which accepts an eSQL statement and returns one or more entities, rows, or scalar values. Each Entity (such as Category) has a set of read/write public properties, which correspond to a table's fields. The setters implement the ReportPropertyChanging and ReportPropertyChanged methods for object value tracking by the ObjectContext. EF's object tracking design is very similar to that for LINQ to SQL's DataContext.

## Modifying Storage to Conceptual Mapping with the EDM Designer

The graphic EDM Designer pane opens with widgets to represent the EntityTypes generated from the \*.edmx file. Only three widgets appear in Figure 1-13; you must drag widgets for the remaining EntityTypes to an appropriate position with regard to the top-level (root) Customers EntityType. Figure 1-13 also shows the Model Browser to the right, which contains tree-views of items in the Northwind.edmx file (NorthwindModel nodes); the Mapping Details pane below the EDM Designer pane lets you modify mapping of fields (columns) to EntityType properties.

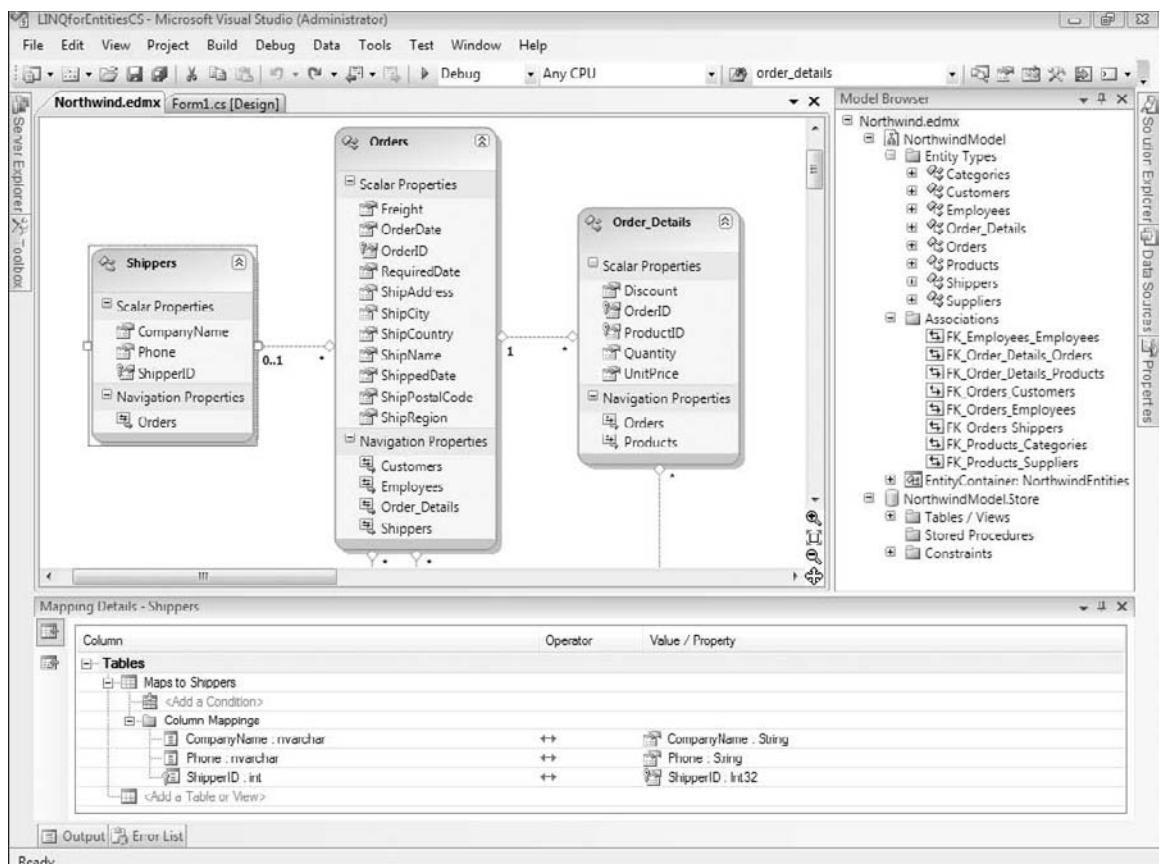


Figure 1-13

## Part I: Getting a Grip on ADO.NET 3.5

---

EntityType properties appear in alphabetic order and foreign key values such as Orders.CustomerID, Orders.EmployeeID, and Orders.Shipper.ID are missing from Figure 1-13's Orders widget. Instead, EDM treats foreign key relationships as Navigation Properties with AssociationSets, which contain bidirectional Associations that specify EntityReferences for many:one and EntitySets for one:many relationships. The differences between LINQ to SQL, which exposes foreign key values and EF, which doesn't, are subtle but add significantly to EF's programming requirements.

*As you learned earlier in the chapter, LINQ to SQL lazy-loads associated entities by default. EF v1 doesn't support deferred loading and you must explicitly request eager loading of associated entities by adding the Extends method to LINQ to Entity queries.*

Unlike LINQ to SQL, EF doesn't singularize EntityType names, so you must change them in the properties sheet for the type or in the widget's text boxes. When you manually singularize EntityType names, such as Orders to Order, EntitySet names automatically gain a ... Set suffix, such as OrderSet. Although you can turn off LINQ to SQL's automatic singularization feature by a property setting in the Option dialog's Database Tools, O/R Designer page, there's no way to inhibit EF v1 from engaging in this nasty behavior. You must revert the EntitySet name to its original plural version by editing the Entity Set Name property in the EntityType's properties sheet.

*It's a good practice to singularize EntityType and navigation property names having 0..1 (Zero or One) multiplicity (many:one relationship) and pluralize EntitySet and navigation property names having \* (Many) multiplicity. The Northwind EDM examples in this book follow this practice.*

*According to the Entity Framework team, adding the Set suffix will be removed or optional in EF v2.*

## **Creating and Binding to a Data Source from a Data Model**

You can create a design-time data source from your EDM by following steps similar to those for LINQ to SQL or typed DataSet data sources. Choose Data, Add New Data Source to open the Data Source Configuration Wizard, select Object in the Choose a Data Source Type dialog, and click Next to open the Select the Object You Wish to Bind To dialog. Expand the project and namespace nodes (both LINQforEntities for this example), select the topmost EntityType you want to bind, Customer for this example (see Figure 1-14), and click Finish to add the new data source. Press F5 to build and run the project, and save your changes.

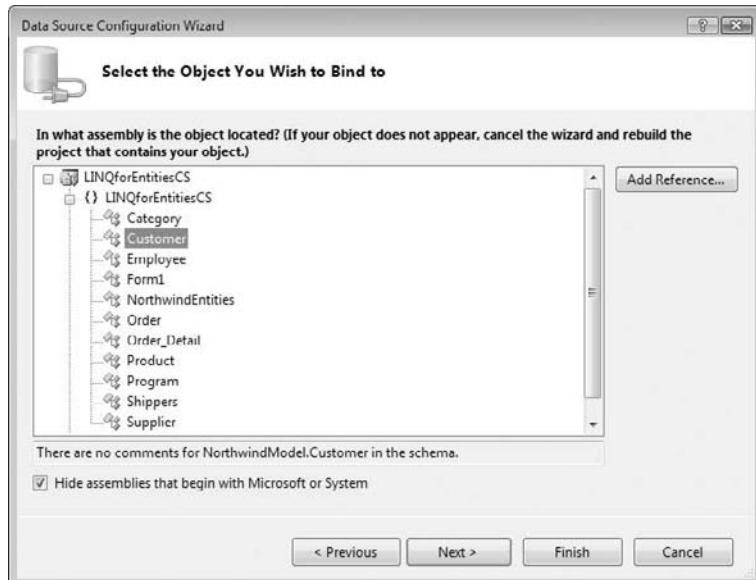


Figure 1-14

The DataSources window displays a tree view of the NorthwindModel object graph (see Figure 1-15). The Orders EntitySet represents a default zero or one:many (0..1 -> \*) Customer.Orders relationship, as illustrated by the end points of the line connecting the Customer and Order widgets. The 0..1 multiplicity for the *one* end of the relationship is incorrect; the Northwind database enforces referential integrity, so you should change the Multiplicity property for the *one* end to 0 in the properties sheet for the AssociationSet.

## Part I: Getting a Grip on ADO.NET 3.5

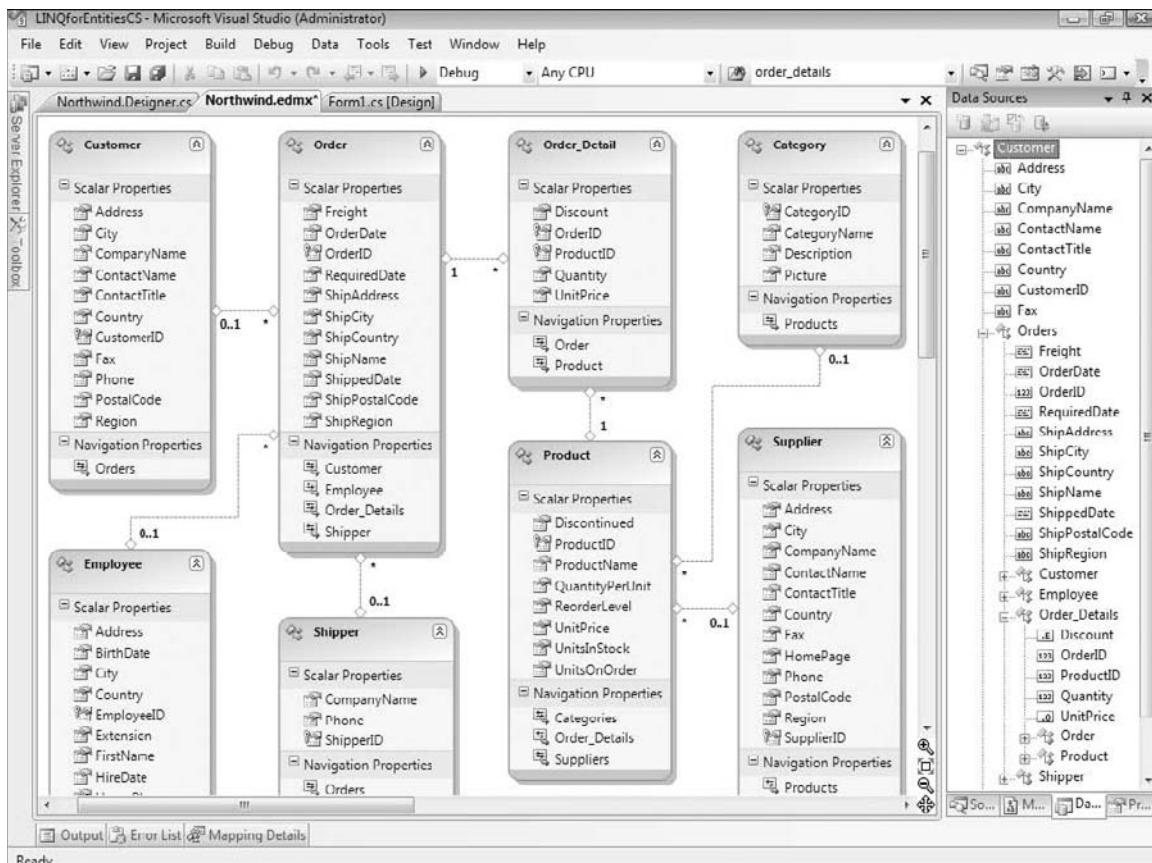


Figure 1-15

To add a BindingNavigator control and BindingSource to the form, as well as bound text boxes to display and edit `Customer` properties, change the Data Sources pane's Customers node from DataGridView to Details and drag it to the form in design view.

## ***Materializing an Object Context***

Dragging the table nodes from Server Explorer to the EDM Designer's surface adds a `<connectionStrings>` group to App.config. Following is the default EF connection string for the Northwind database running under a local instance of SQL Server 2008 Express:

### **XML**

```
<connectionStrings>
  <add name="NorthwindEntities" connectionString="metadata=res://*/Northwind.csdl |
    res://*/Northwind.ssdl|res://*/Northwind.msl; _
    provider=System.Data.SqlClient; provider connection string= _
    &quot;Data Source=localhost\SQLEXPRESS;Initial Catalog=Northwind; _
```

## Chapter 1: Taking a New Approach to Data Access in ADO.NET 3.5

---

```
Integrated Security=True;MultipleActiveResultSets=True";
providerName="System.Data.EntityClient" />
</connectionStrings>
```

*Line continuation characters in the preceding example are inserted to accommodate publishing line length limitations and aren't present in the App.config file.*

The metadata section of the connection string specifies that the three mapping files are stored as a resource (`res:///*/Northwind.*`) in the assembly; the remainder is a standard `SqlClient` connection string and that the `SqlClient` provider has been enhanced to an `EntityClient` provider.

The only code that's required to populate the `Customers` `EntityType` and the text boxes is in the following event handlers:

### C# 3.0

```
private void MainForm_Load(object sender, EventArgs e)
{
    NorthwindEntities ctxNwind = new NorthwindEntities();
    customerBindingSource.DataSource = ctxNwind.Customers;
}
```

### VB 9.0

```
Imports LINQforEntitiesVB

Public Class MainForm
    Private Sub MainForm_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        Dim ctxNwind As New NorthwindEntities
        CustomerBindingSource.DataSource = ctxNwind.Customers
    End Sub
End Class
```

To add bound `DataGridViews` to display the selected `Customer`'s `Orders` and related `Order_Details` `EntitySets`, expand the form and drag the `Customer.Orders` and `Order.Order_Details` nodes from the Data Source pane to the form in design mode.

## Summary

This introductory chapter gave you an overview of the two most important new data-related components of ADO.NET 3.5: Language Independent Query and the Entity Framework.

C# and VB sample code and project examples demonstrated how LINQ increases developer productivity and application reliability for data-intensive projects with these contributions:

- ❑ Query language statements incorporated as first-class members of C# 3.0 and VB 9.0
- ❑ Identical or very similar query syntax for a wide variety of data domains: in-memory objects, relational data, `DataSets`, entities created by EF and EDM, and XML InfoSets (with or without strong typing with XML schemas)

## Part I: Getting a Grip on ADO.NET 3.5

---

- ❑ Full IntelliSense and statement completion for LINQ queries in the VS code editor
- ❑ Strongly typed query statements and results sets to avoid manual casting from `Object`
- ❑ Reduction of application brittleness as the result of compile-time instead of runtime validation of query expressions

The chapter gave you an introduction to programming Microsoft's five LINQ implementations in VS 2008 — LINQ to Objects, LINQ to SQL, LINQ to DataSets, LINQ to Entities, and LINQ to XML. The chapters of Part II deliver the detailed information you need to understand LINQ architecture and the language extensions added to C# 3.0 and VB 9.0 to support the LINQ to Objects implementation. Part III's chapters dig deeply into LINQ to SQL, LINQ to DataSets and LINQ to XML.

Microsoft calls the combination of the ADO.NET Entity Framework, Entity Data Model, and LINQ to Entities *Next Generation Data Access*. You learned that EF and EDM provide a full set of data services based on a unified conceptual schema to create a database-agnostic object persistence layer and object/relational mapping tool with its own LINQ implementation — LINQ to Entities — and SQL-based query language — Entity SQL (eSQL). Microsoft intends to extend the application of EF and EDM to future data reporting, integration, business intelligence, and replication services. This chapter included an overview of the features and benefits of EF; Parts IV and Part V of this book provide the complete coverage.

EF and its components were severed from the RTM version of Visual Studio 2008 and provided in VS 2008 SP1. This chapter's sample code and that of parts IV and V are based on VS 2008 SP1, which you need to run most of the book's code examples.

# **Part II: Introducing Language Integrated Query**

**Chapter 2:** Understanding LINQ Architecture and Implementation

**Chapter 3:** Executing LINQ Query Expressions with LINQ to Objects

**Chapter 4:** Working with Advanced Query Operators and Expressions

## Part II: Introducing Language Integrated Query

---

Chapter 1, “Taking a New Approach to Data Access in ADO.NET 3.0,” gave you a whirlwind tour of Microsoft’s five LINQ implementations included in Visual Studio 2008 and a subsequent Web release: LINQ to Objects, LINQ to SQL, LINQ to DataSets, LINQ to Entities, and LINQ to XML. Chapter 1 emphasized by example the primary benefits of LINQ queries: increased developer productivity and application robustness. LINQ delivers these benefits by enabling strongly typed queries over a variety of data domains with a common set of SQL-like query expressions that are keywords of the C# 3.0 and VB 9.0 languages. Incorporating the elements of query expressions into the programming language eliminates the requirement for brittle strings of SQL statements and, for VB 9.0, XML Infosets.

The chapters of Part II provide the details you need to understand the new concepts, keywords, and syntax that the .NET team added to C# and VB in support of LINQ. Chapter 2, “Understanding LINQ Architecture and Implementation,” covers these new LINQ-related language features in C# 3.0 and VB 9.0:

- ❑ **Local variable type inference** implemented by C# 3.0’s `var` and VB 9.0’s `Dim` keywords to shorten the syntax for declaring and instantiating generic types and support anonymous types
- ❑ **Object initializers** to simplify object construction and initialization with syntax similar to array initializers
- ❑ **Collection initializers** to combine the concept of array initializers and object initializers and extend it to generic collections
- ❑ **Anonymous types** to define inline CLR types without writing a formal class declaration for the type
- ❑ **Lambda expressions** to simplify the syntax of C# 2.0’s anonymous methods, deliver inline functions to Visual Basic developers, and aid in type inference and conversion for expression trees
- ❑ **Extension methods** to enable chaining of extensions that add custom methods to a CLR type without the need to subclass or compile it

In addition to providing support for LINQ, all six new language features let you write more concise and, in most cases, more readable code for development that doesn’t involve LINQ. Writing concise code makes developers more productive, and readable code reduces cost during the later stages of the application life cycle.

*The term concise code in this context doesn’t necessarily mean fewer characters to express the programmer’s intent. If such a criterion were to be applied, VB would lose to C# regardless of the preceding shortcuts. The meaning in this context is fewer noise characters or symbols. Noise represents characters that don’t contribute to achieving the developer’s goal or unnecessary duplication of syntax elements.*

## Functional Languages, Haskell, and Cω

LINQ concepts originated in functional programming languages, such as Haskell (<http://www.haskell.org>), for which *lambda calculus* provides the theoretical underpinnings. (Other well-known functional languages are Excel formulas, Lisp, APL, Erlang, ML, and Scheme.) Imperative programming languages, such as C# and VB, process actions to perform a task in stepwise operations; instead of performing a sequence of actions, functional programs evaluate expressions. Programmers specify what

the program should compute, not how to perform the computation. Haskell is a concise language, to the point of being cryptic even for C# programmers.

SQL is a query language that emulates a functional programming language; an SQL statement defines the rows you want to retrieve or update, and the database engine decides how to perform the requested task. Thus database engines can — and usually do — have very effective query optimizers that are far more effective than sets of instructions written by most mortal programmers.

*Lambda calculus* is a formal mathematical approach to function definition, application, and recursion, and is reified in Haskell and other functional languages as *lambda functions*. Haskell and other functional languages also contributed first-class functions, closures, type inference, lazy evaluation (related to *lazy loading* in LINQ to SQL), monads (a flexible method for combining I/O operations with lazy evaluation) to Microsoft Research's experimental X#, Xen, and finally C $\omega$  (C-Omega) languages (<http://msdn2.microsoft.com/en-us/library/ms974195.aspx>). C $\omega$ 's goal was to reduce the Relational-to-Objects-to-XML (ROX) impedance mismatch by adding concepts and operators from the relational and XML (as defined by the W3C's XML Schema specification) domains into an extension to C#. Microsoft Research released a preview of the C $\omega$  compiler in mid-2004.

C $\omega$  added the following two classes of query expressions to the C# language:

- SQL-based expressions for performing queries with projection, grouping, and joining data from one or more in-memory objects
- XPath-based expressions for querying the members of an XML object by name or by type

The query operators derive from Haskell's *monad comprehensions*, which iterate over a collection, filter out elements that don't meet a specified condition (equivalent to an SQL WHERE clause), and produce a collection by transforming the filtered members. Monad comprehensions, also called *query comprehensions*, provide the foundation for LINQ's query expressions.

Chapter 3, "Executing LINQ Query Expressions with LINQ to Objects" introduces you to query expressions, which some developers call *syntactic sugar* for LINQ's extension methods' explicit syntax for lambda functions. *Syntactic sugar* is an extension to a computer language that makes constructs more succinct, familiar, or both, but doesn't contribute to or detract from the language's functionality.

Chapter 4, "Working with Advanced Query Operators and Expressions," covers more complex SQL-like constructs in LINQ to Objects, such as implementing the equivalent of a LEFT OUTER JOIN, and introduces XPath-based expressions.

Both chapters provide C# 3.0 and VB 9.0 sample code for query expressions and explicit syntax.

## A Brief History of LINQ

A few members of the X#/Xen/C $\omega$  team became associated with the fledgling LINQ group in Anders Hejlsberg's C# fiefdom. Hejlsberg wanted to add to C# a set of "sequence operators" that programmers could apply to collections that implemented the I`Enumerable`<T> generic data type. Eric Meijer (<http://research.microsoft.com/~emeijer>), one of the designers of Haskell98 (whom some call the "creator of LINQ"), became LINQ's primary propagandist on the conference circuit and the author of an extraordinary number of technical papers about current and future LINQ versions. Matt Warren

## Part II: Introducing Language Integrated Query

---

(<http://blogs.msdn.com/mattwar>) made a detour to develop ObjectSpaces, Microsoft's first (and notably unsuccessful) attempt to produce an object/relational mapping (O/RM) tool and then became the architect of LINQ to SQL.

*Matt Warren's blog offers a series of posts that describe the origins and demise of ObjectSpaces. Search his blog with the keyword objectspaces. You can learn more about ObjectSpaces in the introduction to Part IV, "Introducing the ADO.NET Entity Framework."*

LINQ first official preview occurred in September 2005 during Microsoft's Professional Developer's Conference 2005. The LINQ PDC 2005 Technical Preview provided separate C# and VB compilers for Visual Studio 2005 ("Whidbey") Beta 2, which omitted the VB 9.0 version of LINQ to SQL (then called DLinq).

The January 2006 Community Technical Preview (CTP) supported the final (RTM) version of VS 2005, added LINQ to SQL for VB, and improved LINQ IntelliSense features. The OakLeaf Systems blog's "Visual Basic Team Releases January LINQ CTP" post (<http://oakleafblog.blogspot.com/2006/01/visual-basic-team-releases-january.html>) describes the VB release and provides links to early articles and blog posts about LINQ. The C# CTP for the VS 2005 RTM version appeared in December 2005. Delivery of full VB 9.0 implementations of LINQ features consistently trailed C# CTPs by one or more months.

The May 2006 CTP united the C# 3.0 and VB 9.0 implementations with VB still receiving the short end of the features stick. This CTP is considered the reference implementation for VS 2005 with or without the WinFx runtime beta (released as .NET Framework 3.0). A growing group of .NET developers began to climb on the LINQ bandwagon and wrote magazine articles and blogs devoted to or emphasizing LINQ content.

*The ADO.NET team, which was working on the Entity Framework (EF) and Entity Data Model (EDM) independently of the C# LINQ group, announced its Next Generation Data Access vision at Tech\*Ed 2006 in June 2006 and released the first CTP of ADO.NET vNext, which included early versions of EF and EDM, in August 2006. The introduction to Part V offers a brief history of EF, EDM, and LINQ to Entities.*

The Orcas January 2007 CTP included an incomplete implementation of the May 2006 CTP as an add-on to the VS 2008 compiler; LINQ to SQL was missing. The March 2007 CTP, released in late February, was feature-complete for C#, but many VB language constructs for LINQ were still missing. ASP.NET didn't have the LinqDataSource control, and databinding in Windows Forms, especially those that used the Windows Presentation Framework's XAML controls, remained deficient. Orcas Beta 1, which appeared on May 1, 2007, offered no significant improvements to LINQ features. VS 2008 released to manufacturing on November 19, 2007 without EF, EDM, and LINQ to Entities.

VS 2008 SP1 upgraded .NET to v3.5, fixed a few minor bugs in LINQ to Objects, added support for new SQL Server 2008 data types (except GEOGRAPHIC and GEOMETRIC) and performance improvements to LINQ to SQL, and finally released EF, EDM, and LINQ to Entities on August 11, 2008.

# 2

## Understanding LINQ Architecture and Implementation

Most C# and especially VB developers probably have used many *domain-specific programming languages* (commonly called *domain-specific languages* or DSLs) during their careers. DSLs are special-purpose languages dedicated to popular application domains, such as word processing (WordBasic macros), spreadsheets (Excel functions), relational databases (SQL, Embedded Basic, AccessBasic, and Access macros), Web browsers (JavaScript or ECMAScript), music (CSound and hundreds of others), and XML documents (Quilt and XQuery). Visual Basic for Applications (VBA) subsumed WordBasic, AccessBasic, and Excel macros to become the generic DSL for COM-based Windows productivity applications. Microsoft has licensed VBA to hundreds of independent software vendors (ISVs).

DSLs for narrowly specified domains commonly implement code-generating GUIs and wizards. Industry pundits in the late 1990s mistakenly expected DSLs to relieve an impending programmer shortage with automated software factories. VS 2008's LINQ to SQL, Entity Data Model (EDM) Wizard, and EDM Designer, which Chapter 1 introduced, as well as the earlier DataSet designer are special-purpose DSLs that generate C# or VB partial classes from relational databases. Special-purpose DSLs have become so popular that Microsoft offers a set of domain-specific language tools for VS 2005 and later. These tools create custom graphic designers from domain-specific diagram notation and generate code for classes based on text templates.

*DSLs that generate code for small-scale, narrowly defined application components have been quite successful as a whole. However, many enterprise-level developers are loathe to accept "code-gen" and prefer to write their own classes, even when thousands of lines of code are involved, because the generated code doesn't fully conform to organizational or individual developers' standards.*

## Part II: Introducing Language Integrated Query

---

SQL and its dialects, such as SQL Server's Transact-SQL (T-SQL) and Oracle's PL/SQL, as well as Quilt and XQuery are a subclass of DSLs called *domain-specific query languages* or DSQLs. Anders Hejlsberg, Microsoft Distinguished Engineer and chief architect of C#, was interviewed in Jon Udell's "A conversation with Anders Hejlsberg about the May 06 preview of LINQ" podcast ([http://weblog.infoworld.com/udell/gems/ju\\_linqMay06.html](http://weblog.infoworld.com/udell/gems/ju_linqMay06.html)). He said that DSQSLs are "typically very good at queries and . . . really very poor at everything else." Jon Udell observes that DSQSLs are "more bolted on than built in." LINQ is not a DSQSL; it is built into C# 3.0 and VB 9.0.

In the preceding podcast, Anders Hejlsberg describes LINQ as "a unified programming model for querying and transforming and aggregating any kind of data that you would typically manipulate in a data-intensive application, be it object, relational, or XML." That's why LINQ kindled an exceptional amount of interest among .NET developers long before its release for production use in VS 2008.

Although LINQ is not a DSQSL, there are many domain-specific LINQ implementations. As noted in Chapter 1, Microsoft includes five LINQ implementations in VS 2008 SP1:

- ❑ **LINQ to Objects**, the default implementation, for querying in-memory objects
- ❑ **LINQ to SQL** for querying SQL Server [Express], which includes a domain-specific graphic designer for its object/relational mapping (O/RM) tool
- ❑ **LINQ to DataSet** for querying typed and untyped ADO.NET DataSets and standalone DataTables
- ❑ **LINQ to XML** for querying and creating XML Infosets and documents
- ❑ **LINQ to Entities** for querying the Entity Framework's EntitySets defined by the Entity Data Model

*VS 2008 SP1 adds the Entity Framework (EF) and Entity Data Model (EDM) components, which were removed from VS 2008 v1 in late March 2007.*

This chapter covers the underpinnings of LINQ to Objects — primarily new .NET Framework 3.5 namespaces, language extensions, and new compiler features, with code snippets in C# 3.0 and VB 9.0.

## Namespaces That Support LINQ in .NET Fx 3.5

The following table lists the .NET Framework (.NET Fx) 3.5 libraries required to support VS 2008's four standard LINQ implementations. `System.Core`'s LINQ-related extensions enable the incorporation of LINQ to Object features into Silverlight 2 and the Dynamic Language Runtime, which includes `System.Core.dll`.

## Chapter 2: Understanding LINQ Architecture and Implementation

---

Library	Namespace	Members
System.Core.dll	System.Collections.Generic	IEnumerable<T>
	System.Linq	Enumerable, Queryable, and Lookup<TKey, TElement> classes; IGrouping<TKey, TElement>, IOrderedQueryable, IOrderedQueryable<T>, IOrderedSequence<TElement>, IQueryable, and IQueryable<T> interfaces; Func<T, ... TResult> and Action<T, T2>(T1 arg1, T2 arg2,...) delegate functions
	System.Linq.Expressions	Classes, enumerations, and a Funclet delegate function for creating LINQ expression trees
System.Data.Linq.dll	System.Data.Linq	Basic classes, interfaces and enumerations to support LINQ to SQL and its DataContext object
	System.Data.Linq.Expressions	Classes for LINQ to SQL query, insert, update, and delete expressions
	System.Data.Linq.Provider	Provider-agnostic classes, interfaces and enumerations for LINQ to SQL queries and resultsets
	System.Data.Linq.SqlClient	SQL Server 200x-specific classes for LINQ to SQL queries and resultsets
System.Xml.Linq.dll	System.Xml.Linq	Classes and enumerations to define LINQ to XML objects, such as XDocument, XElement, and XAttribute
	System.Xml.Schema	Extension methods for validating XDocument, XElement and XAttribute objects against XML schemas
	System.Xml.XPath	Extension methods for navigating, selecting and evaluating LINQ to XML XNode objects
System.Data._DataSetExtensions.dll	System.Data._DataSetExtensions	EnumerableRowCollection<T> (required for LINQ to DataSet)

## Part II: Introducing Language Integrated Query

---

C# 3.0 and VB 9.0 Windows and Web projects automatically add references to the `System.Core`, `System.Data.DataSetExtensions`, and `System.Xml.Linq` namespaces to support LINQ to Objects, LINQ to DataSet, and LINQ to XML. C# 3.0 adds default `using System.Collections.Generic`, `using System.Linq`, and other `using...` directives to the initial partial class file. VB 9.0 doesn't add any `Imports` directives automatically, so VB developers must add at least the corresponding `Imports System.Collections.Generic` and `Imports System.Linq` directives by hand. Adding a new LINQ to SQL File item from the C# or VB Add New Item dialog's Templates list automatically adds a reference to the `System.Data.Linq` namespace but doesn't add `using/Imports` directives.

## C# and VB Extensions to Support LINQ

LINQ is responsible for most new language extensions in VS 2008. Here's a list of the LINQ-related extensions to C# 3.0 and VB 9.0 with brief descriptions of their purpose:

- ❑ **Implicitly typed local variables** infer the type of local variables from the expressions used to initialize them.
- ❑ **Object initializers** simplify the construction and initialization of objects of arbitrary types.
- ❑ **Array initializers** extend object initializers to elements of arrays of arbitrary types. VB 9.0 doesn't support array initializers.
- ❑ **Collection initializers** (also called implicitly typed array initializers) extend the array initializer style to infer the element type of the collection from the initializer. VB 9.0 doesn't support collection initializers.
- ❑ **Anonymous types** enable creating tuple types with object initializers whose CLR type is inferred by the compiler. C# anonymous types are immutable, but some VB anonymous types are mutable.
- ❑ **Extension methods** enable extending existing and constructed types with new methods.
- ❑ **Lambda expressions** enable C# 2.0's anonymous methods to deliver better type inference and allow conversions to expression trees and delegate types. VB 9.0 supports Lambda expressions but doesn't expose anonymous methods to programmers.
- ❑ **Standard Query Operators** (SQOs. Also called *Standard Sequence Operators*) filter, transform, join, group, and aggregate collections of CLR types.
- ❑ **Query expressions** group SQOs into a high-level query language that is similar to SQL and related to XQuery.
- ❑ **Expression trees** permit lambda expressions to be represented as data (expression trees) instead of as code (delegates).
- ❑ `IQueryable<T>` and `IOrderedQueryable<T>` interfaces, in conjunction with expression trees, facilitate third-party LINQ implementations for other data domains and enable the compiling of queries for improved performance. `IQueryable<T>` and `IOrderedQueryable<T>` types enable composable queries by passing them as the data source of one or more additional LINQ queries.

LINQ also relies on these language features of C# 2.0 and VB 8.0:

- ❑ **Generic types** let you specify the precise data type that a method, class, structure, or interface acts upon. The primary benefits of generics are type safety and increased code reusability.
- ❑ **Anonymous methods** (also called anonymous delegates) let you define a nameless method that's called by a delegate and access variables that would normally be out of scope. VB doesn't support anonymous methods directly.
- ❑ **Iterators**, specifically the generic, type-safe `IEnumerable<T>` interface, in the `System.Collections.Generic` namespace, which supports iterating over a generic collection in a `foreach` or `For Each...Next` loop.

Programs that don't execute LINQ queries can take advantage of most new VS 2008 features. LINQ will introduce the more abstract functional constructs, such as lambda expressions and expression trees, to most .NET developers.

### ***Implicitly Typed Local Variables***

The new C# 3.0 `var` keyword declares an implicitly typed local variable; this feature also is known as local variable type inference (LVTI), as used in Chapter 1. Implicitly typed local variables infer their type from the initializer expression to the right of a local declaration statement's `=` symbol. This table compares declarations for C# 3.0 implicitly typed local variables with explicitly typed declarations in C# 2.0 and earlier:

C# 3.0 Implicitly Typed Declaration	C# 2.0 Explicitly Typed Declaration
<code>var Quantity = 30;</code>	<code>int Quantity = 30;</code>
<code>var QuantityPerUnit = "12 1-kg cartons";</code>	<code>string QuantityPerUnit = "12 1-kg cartons";</code>
<code>var UnitPrice = 15.55;</code>	<code>double UnitPrice = 15.55;</code>
<code>var OrderDate = new DateTime(2008, 1, 5);</code>	<code>DateTime OrderDate = new DateTime(2008, 1, 5);</code>
<code>var ShippedDate = DateTime.Now;</code>	<code>DateTime ShippedDate = DateTime.Now;</code>
<code>var Discontinued = false;</code>	<code>bool Discontinued = false;</code>
<code>var numbers = new int[] { 0, 1, 2, 3, 4 };</code>	<code>Int[] numbers = new int[] { 0, 1, 2, 3, 4 };</code>

For numeric types, `var` infers the most common type: `int` or `Int32` for whole numbers and `double` for numbers with decimal fractions.

*C# sample code for this chapter is behind the `Extensions.cs` form of the \WROX\ADONET\Chapter02\CS\Extensions\ExtensionsCS.sln project, except as noted.*

## Part II: Introducing Language Integrated Query

VB 9.0 substitutes Dim for var in the following implicitly typed declarations:

VB 9.0 Implicitly Typed Declaration	VB 8.0 Explicitly Typed Declaration
Dim Quantity = 30	Dim Quantity As Integer = 30
Dim QuantityPerUnit = "12 1-kg cartons"	Dim QuantityPerUnit As String = "12 1-kg cartons"
Dim UnitPrice = 15.55	Dim UnitPrice As Double = 15.55
Dim OrderDate = #1/1/2008#	Dim OrderDate As Date = #1/1/2008#
Dim ShippedDate = DateTime.Now	DateTime ShippedDate = DateTime.Now
Dim Discontinued = False	Dim Discontinued As Boolean = False
Dim numbers = New Integer() { 0, 1, 2, 3, 4 };	Dim numbers As Integer() = _New Integer() { 0, 1, 2, 3, 4 };

VB 9.0 initializes whole numbers as Integer and those with decimal fractions as Double.

*VB sample code for this chapter is behind the Extensions.vb form of the \WROX\ADONET\Chapter02\VB\Extensions\ExtensionsVB.sln project, except as noted.*

Implicitly typed local variables don't save a substantial amount of typing, especially in C#, but they're needed when you deal with anonymous types or don't know in advance what the type of a dynamically generated local type will be.

## Object Initializers

Object initializers save on typing by reducing the initialization code for a complex object to a single expression that doesn't require prefixing C# property names with the object name. For example, consider creating an instance of the following LineItem class, which uses a new C# 3.0 automatic properties shortcut for creating default getters and setters:

### C# 3.0

```
public class LineItem
{
    // Automatic properties
    public int OrderID { get; set; }
    public int ProductID { get; set; }
    public short Quantity { get; set; }
    public string QuantityPerUnit { get; set; }
    public decimal UnitPrice { get; set; }
    public float Discount { get; set; }
}
```

## Chapter 2: Understanding LINQ Architecture and Implementation

---

The traditional C# approach to creating an instance of `LineItem` is:

### C# 1.0 and 2.0

```
LineItem line2 = new LineItem();
    Line2.OrderID = 11000;
    line2.ProductID = 61;
    line2.Quantity = 30;
    line2.QuantityPerUnit = "12 1-kg cartons";
    line2.UnitPrice = 15.55M;
    line2.Discount = 0.15F;
```

The `M` and `F` suffixes specify `decimal` and `float` data types, respectively.

Here's the C# 3.0 expression-based object initialization code:

### C# 3.0

```
var line3 = new LineItem { OrderID = 11000, ProductID = 61, Quantity = 30,
    QuantityPerUnit = "12 1-kg cartons", UnitPrice = 15.55M, Discount = 0.15F };
```

If publication requirements didn't require splitting the expression, the object initialization statement could appear as a single instruction on a single line. Expression-based object initialization is critical to creating *anonymous types* for LINQ queries, which you define with `var query = from...select...` expression or `Dim Query = From...Select...expression` instructions. Anonymous types are the subject of a section that follows shortly.

The traditional VB `With...End With` syntax for initializing the corresponding `LineItem` object is:

### VB 8.0 and earlier

```
Dim Itm8 As New LineItem
With Item8
    .OrderID = 11000
    .ProductID = 61
    .Quantity = 30
    .QuantityPerUnit = "12 1-kg cartons"
    .UnitPrice = 15.55D
    .Discount = 0.15F
End With
```

VB uses the `D` suffix for the `Decimal` data type.

VB 9.0 expression-based initialization code is a minor modification of the `With...End With` structure. Here's VB 9.0 code that's compatible with LINQ `Select` expressions:

### VB 9.0

```
Dim Itm9 As New LineItem With {.OrderID = 11000, .ProductID = 61, .Quantity = 30,
    .QuantityPerUnit = "12 1-kg cartons", .UnitPrice = 15.55D, .Discount = 0.15F}
```

## Part II: Introducing Language Integrated Query

---

### **Array Initializers with Object Initializers**

Array initializers let you declare and add multiple elements to an array of the object type with object initializer syntax. The syntax is as follows for C# 3.0 LineItem instances:

#### **C# 3.0**

```
var LineItems = new[]
{
    new LineItem {OrderID = 11000, ProductID = 11, Quantity = 10,
        QuantityPerUnit = "24 500-g bottles", UnitPrice = 15.55M, Discount = 0.0F},
    new LineItem {OrderID = 11000, ProductID = 21, Quantity = 20,
        QuantityPerUnit = "12 1-kg cartons", UnitPrice = 20.2M, Discount = 0.1F},
    new LineItem {OrderID = 11000, ProductID = 31, Quantity = 30,
        QuantityPerUnit = "24 1-kg bags", UnitPrice = 25.45M, Discount = 0.15F}
};
```

As noted earlier, VB 9.0 object initializers can't be used to initialize arrays. However, you can assign elements created by object initializers to an array declared previously with this workaround:

```
Dim LineItems(2) As LineItem
LineItems(0) = New LineItem With {.OrderID = 11000, .ProductID = 11, _
    .Quantity = 10, .QuantityPerUnit = "24 500-g bottles", .UnitPrice = 15.55D, _
    .Discount = 0.0F}
LineItems(1) = New LineItem With {.OrderID = 11000, .ProductID = 21, _
    .Quantity = 20, .QuantityPerUnit = "12 1-kg cartons", .UnitPrice = 20.2D, _
    .Discount = 0.1F}
LineItems(2) = New LineItem With {.OrderID = 11000, .ProductID = 31,
    .Quantity = 30, .QuantityPerUnit = "24 1-kg bags", .UnitPrice = 25.45D, _
    .Discount = 0.15F}
```

### **Collection Initializers**

Collection initializers are the corollary of array initializers for collections that support `IEnumerable` and have an `.Add` method. For example, here's the C# code to create and populate a generic `List<LineItem>`:

#### **C# 3.0**

```
var LineItemsList = new List<LineItem>
{
    new LineItem {OrderID = 11000, ProductID = 11, Quantity = 10,
        QuantityPerUnit = "24 500-g bottles", UnitPrice = 15.55M, Discount = 0.0F},
    new LineItem {OrderID = 11000, ProductID = 21, Quantity = 20,
        QuantityPerUnit = "12 1-kg cartons", UnitPrice = 20.2M, Discount = 0.1F},
    new LineItem {OrderID = 11000, ProductID = 31, Quantity = 30,
        QuantityPerUnit = "24 1-kg bags", UnitPrice = 25.45M, Discount = 0.15F}
};
```

## Chapter 2: Understanding LINQ Architecture and Implementation

---

Here's the workaround to create and initialize a VB 9.0 collection with object initializers and the `List(Of T).Add` method:

```
Dim LineItemsList As New List(Of LineItem)
With LineItemsList
    .Add(New LineItem With {.OrderID = 11000, .ProductID = 11, .Quantity = 10, _
        .QuantityPerUnit = "24 500-g bottles", .UnitPrice = 15.55D, .Discount = 0.0F})
    .Add(New LineItem With {.OrderID = 11000, .ProductID = 21, .Quantity = 20, _
        .QuantityPerUnit = "12 1-kg cartons", .UnitPrice = 20.2D, .Discount = 0.1F})
    .Add(New LineItem With {.OrderID = 11000, .ProductID = 31, .Quantity = 30, _
        .QuantityPerUnit = "24 1-kg bags", .UnitPrice = 25.45D, .Discount = 0.15F})
End With
```

## Anonymous Types

Anonymous types, also called *projections*, are an abbreviated form of object initializers that let you omit the type specification — for example, `LineItem` in the earlier *Object Initializers* section's code, when initializing temporary objects or collections. If you can omit the type specification, it's clear that you can drop the corresponding (`LineItem`) class definition also. Anonymous types are essential to LINQ projections, which let you incorporate the LINQ equivalent of an SQL field list in the query's `select`/`Select` or `selectMany`/`SelectMany` clause.

*Chapter 1's Sample LINQ Queries section introduced you to projections.*

Projections define *data shapes* in terms of *tuples*. A tuple is the mathematical term for a finite series of objects of a particular type, in this case the elements you specify for your output collection. How you combine related fields from multiple data sources, such as a relational table and an XML InfoSet, determines the *data shape*. For example, consider the anonymous type specified by the `select` clause of the following C# query:

### C# 3.0

```
var query = from i in LineItems
    select new { i.OrderID, i.ProductID, i.UnitPrice }
```

The simple data shape specified by the emphasized anonymous type is a *3-tuple* (*triple* in ordinary English); anonymous types use *n-tuple* terminology. The query variable is of the `IEnumerable<3-Tuple>` type.

Here's the VB 9.0 version, which doesn't need the `New` keyword and French braces if each tuple's suffix is a property name:

### VB 9.0

```
Dim Query = From i In LineItems
    Select i.OrderID, i.ProductID, i.UnitPrice
```

The compiler infers the type from the tuples, so the resulting object is strongly typed. The compiler treats multiple instances of an object that has the same names and types in the same order as being of the same type. The compiler reports the C# example's `query` type as `IEnumerable<<int, int, decimal>>` and the VB example's `Query` type as `IEnumerable(Of VB$AnonymousType_0(Of`

## Part II: Introducing Language Integrated Query

---

`Integer, Integer, Decimal)).` If the values of two instances are equal also, the two instances have the same hashcodes compiler considers the two instances equal.

As mentioned earlier, C# 3.0 anonymous types are immutable; their property constructors have getters only. VB anonymous types are mutable, but you can use the `Key` modifier to specify read-only tuples (fields) to be included in the instance's hash value as in this example, where `OrderID` and `ProductID` form a composite key:

```
New With { Key .OrderID, Key .ProductID, .UnitPrice }
```

Maintaining mutability makes anonymous types more useful for applications that don't involve LINQ.

## Extension Methods

Extension methods let you add custom methods to previously defined types, even if those types are sealed in C# or `NotInheritable` in VB. For example, the `String /String` class is sealed and has 45 predefined instance methods, which you access by dot (.) syntax, as in `strTest.Length()` or `"This is a test".Length()`. The compiler treats these statements as if they were members of a function library: `Length(strTest)` or `Length("This is a test")`.

The `Length` method throws an exception if the class value is `null` or `Nothing`, so a `LengthNullable()` method that returns an `int?, Integer?, or Nullable(Of Integer)` type eliminates the need to check for `null` or `Nothing` before invoking the `Length` method. Here's the C# 3.0 code for the `LengthNullable` method:

### C# 3.0

```
static class ExtensionMethods
{
    public static Int32? LengthNullable(this string test)
    {
        if (test != null)
        {
            return test.Length;
        }
        else
        {
            return null;
        }
    }
}
```

*Sample code for the C# 3.0 ExtensionMethods class is located in the ExtensionMethods.cs class file.*

## Chapter 2: Understanding LINQ Architecture and Implementation

According to VS 2008's online help's "extension methods [Visual Basic]" topic, you can apply extension methods to any of the following types:

- Classes (reference types)
- Structures (value types)
- Interfaces
- Delegates
- ByRef and ByVal arguments
- Generic method parameters
- Arrays

C# 3.0 extension methods must be defined by a `public static` method within a `public static` or `internal static` class. Prefixing the method's first argument with the `this` keyword (highlighted in the code) informs the compiler that the function is an extension method. The second argument is the type to which the method applies, `string` for this example, and the third is the instance name, `test`. Extension methods support full statement completion and IntelliSense features, as shown in Figure 2-1.

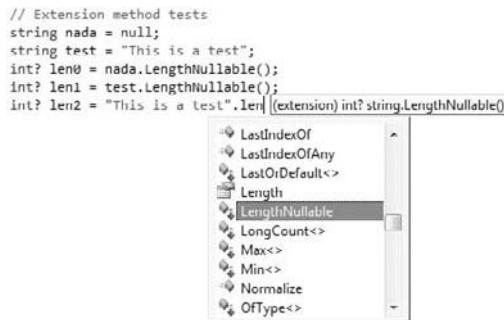


Figure 2-1

A downward-pointing arrow added to the IntelliSense list's method icon identifies extension methods. Following are a few C# test methods to verify that the new extension method behaves as expected:

### C# 3.0

```
// Import the namespace if method is in another project
using CS3Extensions.ExtensionMethods;

// Extension method tests
string nada = null;
string test = "This is a test";
int? len0 = nada.LengthNullable();
int? len1 = test.LengthNullable();
int? len2 = "This is a test".LengthNullable();
```

## Part II: Introducing Language Integrated Query

---

If you create a C# class library for your extension methods, you must import the namespace with a `using` directive. (Namespace importation isn't required for the sample code because it's in the same project, if the class in the library has a namespace that isn't already in scope from the referenced library.)

If an extension method has the same signature as an instance method, the call will invoke the instance method. VS 2008's online help's "extension methods [Visual Basic]" topic states that the compiler invokes extension methods with the following precedence:

1. Extension methods defined inside the current module (if there is one).
2. Extension methods defined inside data types in the current namespace or any one of its parents, with child namespaces having higher precedence than parent namespaces.
3. Extension methods defined inside any type imports in the current file.
4. Extension methods defined inside any namespace imports in the current file.
5. Extension methods defined inside any project-level type imports.
6. Extension methods defined inside any project-level namespace imports.

You can create VB 9.0 extension methods in modules only, and you must add an `Imports System.Runtime.CompilerServices` directive to support the `<Extension()>` attribute. This attribute is necessary to instruct the compiler that the function is an extension method. VB functions declared within modules are static, so the `Shared` keyword isn't required. The argument's type declaration specifies the class to which the extension method applies.

### VB 9.0

```
Imports System.Runtime.CompilerServices

Public Module ExtensionMethods
    ' VB extension methods can only be declared within modules
    <Extension()>
    Public Function LengthNullable(ByVal Test As String) As Integer?
        If Test Is Nothing Then
            Return Nothing
        Else
            Return Test.Length
        End If
    End Function
End Module
```

*Sample code for the VB 9.0 ExtensionMethods module is located in the ExtensionMethods.vb module file.*

As with the C# version, you must import the VB module's namespace to bring the extension method within the application's scope, if the module isn't in the same project. The VB test code is quite similar to that for the C# version.

### VB 9.0

```
' Namespace required if the module is in another project
' Imports ExtensionsVB.ExtensionMethods

' Extension methods tests
Dim strNull As String = Nothing
Dim strTest As String = "This is a test"
Dim intLen0 As Integer? = strNull.LengthNullable
Dim intLen1 As Integer? = strTest.LengthNullable
```

*The Type? shortcut for Nullable(Of Type) is new in VB 9.0.*

*VB developers probably would use the Len() function from the Microsoft.VisualBasic.Strings module instead of creating an extension method because the Len() function returns 0 for a null string argument.*

## Anonymous Methods and Generic Predicates

C# 2.0 introduced anonymous methods, but few C# developers took advantage of them because of their somewhat strange syntax. The primary use of anonymous methods is to eliminate the need to create a simple method in situations where only a delegate is sufficient for the task. A delegate is a type that stores a reference (a pointer in C++) to a method. In earlier C# versions, you had to use named a method to declare a delegate with pseudocode like this:

### C# 2.0

```
// Declare a delegate
delegate void Delegate(int x);

// Define a named method
void DoSomething(int y) { /* Something */ };

// Declare the delegate using the method as a parameter
Delegate d = obj.DoSomething;
```

An anonymous method lets you define a nameless method that's called by a delegate, as in the following pseudocode:

### C# 2.0

```
// Create a delegate instance
delegate void Delegate(int x);

// Instantiate the delegate using an anonymous method
Del d = delegate(int y) { /* Do Something */ };
```

Anonymous methods enable writing cleaner, more compact code and consume fewer resources than named methods. Anonymous methods also have the unique capability to access local variables of the method in which they're declared. Because anonymous methods are closely associated with delegates, they also are known as *anonymous delegates*.

## Part II: Introducing Language Integrated Query

---

C# 2.0 and VB 8.0 introduced `System.Collections.Generic.List<T>` and a update to `System.Array` that include methods for testing (`Exists()`), finding (`Find()`, `FindAll()`, `FindIndex()`, `FindLast()`, `FindLastIndex()`) and, for `List<T>`, removing (`RemoveAll()`) elements. The methods enable simple queries over arrays and generic lists. These methods have the following signatures and, for VB, usage:

### C# 2.0

```
public [static] T Find<T> (
    T[] array,
    Predicate<T> match
)
```

### VB 8.0

```
Public [Shared] Function Find(Of T) ( _
    array As T(), _
    match As Predicate(Of T) _
) As T

' Usage
Dim array As T()
Dim match As Predicate(Of T)
Dim returnValue As T

returnValue = Array.Find(array, match)
```

The `static/Shared` declarations apply to the `Array` type not `List<T>`. Each of these methods takes a `Predicate<T>/Predicate(Of T)` generic delegate to provide the criterion with which to test, find, or remove elements. Here are the signatures and, for VB, usage of generic delegates:

### C# 2.0

```
public delegate bool Predicate<T> (
    T obj
)
```

### VB 8.0

```
Public Delegate Function Predicate(Of T) ( _
    obj As T _
) As Boolean

' Usage
Dim instance As New Predicate(Of T)(AddressOf HandlerMethod)
```

## Chapter 2: Understanding LINQ Architecture and Implementation

---

Following are examples of the use of the named `HighUnitPrice()` method with generic delegates to find the first `LineItem` element in a `LineItems` array whose `UnitPrice` is more than \$25.00:

### C# 2.0

```
// Predicates test (named method)
LineItem obj = Array.Find(LineItems, HighUnitPrice);

static bool HighUnitPrice(LineItem i)
// For Predicates test (named method)
{
    if (i.UnitPrice > 25M)
        return true;
    else
        return false;
}
```

### VB 8.0

```
' Predicates test (named method)
Dim objFirst As LineItem = Array.Find(LineItems, AddressOf HighUnitPrice)

Private Shared Function HighUnitPrice(ByVal i As LineItem) As Boolean
    ' For Predicates test (named method)
    If i.UnitPrice > 25D Then
        Return True
    Else
        Return False
    End If
End Function
```

C# 3.0 lets you combine anonymous methods with anonymous types into a considerably simpler pair of statements, shown emphasized in the following example:

### C# 3.0

```
// Predicates test (anonymous method)
var anon = LineItemsList.Find( delegate(LineItem i)
    { return i.UnitPrice >= 25M; } );
```

Lambda expressions further simplify the anonymous method syntax and implement the standard query operators for LINQ query expressions, as you'll see in the next section.

## **Lambda Expressions**

A lambda expression is a convenient means to assign an expression or block of code (an anonymous method) to a variable (a delegate). A lambda expression accomplishes this with an absolute minimum amount of typing. Although VB 9.0 doesn't offer anonymous methods, it does support lambda expressions, which members of Microsoft's VB team also call *inline methods*. Without support for lambda methods VB developers couldn't take advantage of LINQ.

## Part II: Introducing Language Integrated Query

---

Here's the basic syntax for lambda expressions:

```
argument-list => expression-or-statement-block // C# 3.0  
Function(argument-list) expression ' VB 9.0
```

Lambda expressions extend anonymous methods with the following capabilities:

- Parameter type inference, which eliminates the need to specify parameter types
- Conversion to expression trees, which treat code as data
- Pass as method arguments, implementing argument type inference and method overload resolution
- Substitution of C# statement blocks for expressions, where necessary

*VB 9.0 lambda expressions don't support statement blocks.*

Lambda expressions implement the method call syntax form of LINQ's Standard Query Operators (SQO), which are the subject of the later "Standard Query Operators" section and Chapter 3 and Chapter 4.

### C# 3.0 Lambda Expressions

Moving from the anonymous method to lambda expression syntax is a stepwise process. Here are the steps using the sample `delegate(LineItem i) { return i.UnitPrice >= 25M; }` anonymous method from the last (C#) example of the earlier "Anonymous Methods and Generic Predicates" section:

1. Delete the `delegate` keyword, leaving `(LineItem i) { return i.UnitPrice >= 25M; }`.
2. Replace the French braces with the lambda operator ("fat arrow"), leaving `(LineItem i) => return i.UnitPrice >= 25M;`.
3. Remove the `return` keyword and delete the semicolon because `i.UnitPrice >= 25M` has become an expression, leaving `(LineItem i) => i.UnitPrice >= 25M.`
4. The compiler can infer the type, so remove `LineItem` and the parentheses, which leaves `i => i.UnitPrice >= 25M`, an operable lambda expression to supply the predicate.

The lengthy predicates test code that uses an anonymous function becomes:

#### C# 3.0

```
// Predicates test (lambda function)  
var anon = LineItemsList.Find( i => i.UnitPrice >= 25M );
```

The `Find()` method of the preceding example is called a *singleton method* because it returns a single `LineItem` type, not an `IEnumerable<T>` type.

*Most folks pronounce the lambda operator (=>) "goes to" but some prefer "such that" or "begets."*

Generic `List<T>` types support `IEnumerable<T>` and `foreach/ForEach` iteration. The following code, which resembles some of the simple LINQ queries you saw in Chapter 1, adds two `LineItemsList` rows and a count of `LineItems` with `UnitPrice` values  $\geq \$20.00$ .

### C# 3.0

```
// Predicates test (lambda function, multiple values)
result = "\r\nPredicate(Of T) Test (Lambda Function, Multiple Items) \r\n";
var mult = LineItemsList.FindAll(i => i.UnitPrice >= 20M);
foreach (var i in mult)
    result += i.OrderID.ToString() + "\t" + i.ProductID.ToString() + "\t\t" +
        i.UnitPrice.ToString() + "\r\n";
txtResult.Text += result;
```

The `FindAll()` method returns an `IEnumerable<T>` type to enable iteration with the `foreach` operator. The `FindAll()` method isn't invoked against the `LineItemsList` collection until execution reaches the `foreach` operator, which executes yield return.

Another capability of lambda expressions is composability by chaining extension methods. Here's a simple example that returns the number of items with a `UnitPrice` value  $\geq \$20$ :

### C# 3.0

```
var quan = LineItemsList.FindAll(i => i.UnitPrice >= 20M).Count;
txtResult.Text += quan.ToString() + " Items with Price >= $20.00\r\n";
```

You can chain multiple lambda expressions that return `IEnumerable<T>` types to apply complex filtering and sorting operations to arrays and `List<T>` types. The following “Standard Query Operators” section provides examples of chained lambda expressions.

## VB 9.0 Lambda Expressions

VB 9.0 lambda expressions differ from their C# 3.0 counterparts in syntax but otherwise behave identically. You create a VB 9.0 lambda expression in the `Function(argument-list)` expression format from a conventional VB function by removing the function name, `ByVal`, return type, the `Return` keyword, and the `End Function` (function footer) statement. For example, the lambda form of the `HighUnitPrice()` function in the earlier “Anonymous Methods and Generic Predicates” section becomes:

```
Function(i As LineItem) i.UnitPrice >= 20D
```

Argument types are optional; omitting the argument type (`As LineItem`) creates an *implicitly typed lambda expression*. Most lambda expressions are implicitly typed.

The following VB 8.0 snippet uses a named method to return an array of `LineItems` to the iterator:

### VB 8.0

```
Dim objItems() = Array.FindAll(LineItems, AddressOf MidUnitPrice)

For Each i In objItems
    strResult &= i.OrderID.ToString & vbTab & i.ProductID.ToString & vbTab & _
        i.UnitPrice.ToString & vbCrLf
Next
```

## Part II: Introducing Language Integrated Query

---

This code uses an implicitly typed lambda expression (emphasized) to return the same array to the iterator:

### VB 9.0

```
Dim objItems() = Array.FindAll(LineItems, Function(i) i.UnitPrice >= 20D)

For Each i In objItems
    strResult &= i.OrderID.ToString & vbTab & i.ProductID.ToString & vbTab & _
        i.UnitPrice.ToString & vbCrLf
Next
```

## Standard Query Operators

The `System.Linq.Enumerable` static class declares a set of methods known as the Standard Query Operators (SQOs). LINQ SQOs are the primary subject of the next chapter but deserve a brief discussion here because of their close relationship with lambda expressions and expression trees. The next section covers expression trees.

Most SQOs are static extension methods that extend `IEnumerable<T>` to the point where it becomes a complete query language for arrays and collections that implement `IEnumerable<T>`. The most popular extension methods are `Where`, `OrderBy`, and `Select`, which you saw in most Chapter 1 examples. Most of Chapter 3 is devoted to LINQ's default SQOs implementation by LINQ to Objects.

*Names of extension methods are PascalCase, despite their corresponding C# keywords being lower case. VB 9.0's keyword for OrderBy is Order By, presumably to more closely resemble SQL; otherwise VB preserves the methods' original PascalCase nomenclature.*

The `from/From` keywords don't represent extension methods and aren't SQOs; `from` and `From` specify a range variable or alias in a sequence whose name follows the `in/In` operators.

## C# 3.0 Query Expressions

All C# 3.0 query expressions start with one or more `from aliasName in sequenceName` clause and end with either a `select` or `group` clause. Optional `join`, `let`, `where`, and `orderby` clauses can appear between the last `from` and the `select` or `group` clause.

*Query expressions are also known as query comprehensions, which is a name derived from the Haskell language's list (or monad) comprehensions. Haskell's list comprehensions can be written as expressions that include the higher-order functions map and filter to create one list from another. In-memory object collections, relational database tables, and XML Infosets are types of lists.*

## Chapter 2: Understanding LINQ Architecture and Implementation

---

Here's an example of a simple `select` query expression against an in-memory `List<productList>`, which generates a list of the `ProductID`, `Category`, and `ProductName` values for out-of-stock products:

### C# 3.0

```
var noStock = from p in productList
              where p.UnitsInStock == 0
              orderby p.Category, p.ProductID
              select new { p.ProductID, p.Category, p.ProductName };

foreach (var i in noStock)
    result += i.ProductID.ToString() + "\t" + i.Category.Substring(0, 6) + "\t" +
    i.ProductName + "\r\n";
```

The compiler translates the preceding query expression into the following chain of explicit extension method calls, which take lambda expressions as their argument:

### C# 3.0

```
var noStock = productList
    .Where(p => p.UnitsInStock == 0)
    .OrderBy(p => p.Category)
    .ThenBy(p => p.ProductID)
    .Select(p => new { p.ProductID, p.Category, p.ProductName });
```

The `Where` extension method processes the `IEnumerable<productList>` and passes the filtered result to the `OrderBy` and `ThenBy` methods, and finally to the `Select` method, which trims the members sent to the `foreach` iterator. The `System.Linq.Enumerable` class defines the basic set of LINQ extension methods required to implement LINQ to Objects.

Here's the C# source code for the `Where` extension method, which receives the `p => p.UnitsInStock == 0` as the source (`IEnumerable<p>`) and predicate (`p, bool UnitsInStock == 0`) arguments:

### C# 3.0

```
namespace System.Linq {
    using System;
    using System.Collections.Generic;

    public static class Enumerable {
        public static IEnumerable<T> Where<T>(
            this IEnumerable<T> source,
            Func<T, bool> predicate) {

            foreach (T item in source)
                if (predicate(item))
                    yield return item;
        }
    }
}
```

## Part II: Introducing Language Integrated Query

---

`Func<T, bool>` is a delegate type that's provided with several overloads by the `System.Linq.Enumerable` class. You can use `Func<T, bool>` to represent any function that takes a single parameter and returns a `bool` type.

*VB 9.0 doesn't support C# 2.0+'s `yield` keyword. Converting the preceding example to VB would require implementing a state machine.*

The most important element of the `Where` extension method code is the `yield return item` instruction that the iterator executes if `item` meets the `Where` criterion (`predicate expression`). C# 2.0 introduced the `yield` contextual keyword but few developers used it. The compiler recognizes `yield` as a keyword only if it's followed by `return` or `break`. The `yield return item` instruction in an iterator block enables an enumerable class to be implemented in terms of another enumerable class; that is, chained.

### **Lazy and Eager Evaluation**

The `yield return item` instruction enables *lazy evaluation*, which is another feature critical to LINQ. Lazy evaluation means that a LINQ code doesn't touch the query's data source until the code executes the iterator's first `yield return item` instruction. Lazy evaluation makes execution of chained extension methods very efficient by eliminating storage of intermediate results. Lazy evaluation is the key to object/relational mapping (O/RM) tools' *lazy loading* of objects related to your main object by associations. Lazy loading means that the O/RM tool doesn't load related objects until the application needs to access their property values. Chapter 5 goes deeper into LINQ to SQL's approach to lazy loading.

*Eager evaluation* means that the query is self-executing. For example, chaining an aggregate function to the end of a query causes immediate evaluation. The following snippets demonstrate eager evaluation with the `Count()` expression method in query expression and method call syntax:

#### **C# 3.0**

```
var noStockCount1 = (from p in productList
                     where p.UnitsInStock == 0
                     select p).Count();
result = "\r\nCount of out-of-stock items = " + noStockCount1.ToString();

var noStockCount2 = productList
                     .Where(p => p.UnitsInStock == 0)
                     .Count();
result = "\r\nCount of out-of-stock items = " + noStockCount2.ToString();
```

#### **VB 9.0**

```
Dim NoStockCount1 = (From p In productList
                     Where p.UnitsInStock = 0
                     Select p).Count();
Result = vbCrLf & "Count of out-of-stock items = " & NoStockCount1.ToString();

Dim NoStockCount2 = productList
                     .Where(Function(p) p.UnitsInStock = 0)
                     .Count();
Result = vbCrLf & "Count of out-of-stock items = " & NoStockCount1.ToString();
```

### VB 9.0 Query Expressions

VB 9.0 query expressions differ only slightly from corresponding C# 3.0 versions. Here's the VB version of the first query expression of the earlier "C# 3.0 Query Expressions" section:

#### VB 9.0

```
Dim NoStock = From p In productList _
    Where p.UnitsInStock = 0 _
    Order By p.Category, p.ProductID _
    Select p.ProductID, p.Category, p.ProductName

For Each i In NoStock
    strResult &= i.ProductID.ToString() & vbTab & i.Category.Substring(0, 6) & _
        vbTab & i.ProductName & vbCrLf
Next
```

And here's the equivalent chained extension method call syntax:

#### VB 9.0

```
Dim NoStock = productList
    .Where(Function(p) p.UnitsInStock = 0)
    .OrderBy(Function(p) p.Category)
    .ThenBy(Function(p) p.ProductID)
    .Select(Function(p) p.ProductID, p.Category, p.ProductName )
```

## Expression Trees and Compiled Queries

The C# or VB compiler can convert lambda expressions to expression trees, which enable treating code as a data structure stored in an `Expression<T>` type. Representing code as data lets programs modify the code and interpret or compile it for reuse. LINQ to SQL uses expression trees to translate the method (chained lambda expressions) form of the query expression to T-SQL, which is sent to the SQL Server instance when execution reaches the `foreach/For Each` block or an aggregate expression. Expression trees can substitute for any of the SQOs defined by the `System.Linq.Queryable` class, which are the same as those for `System.Linq.Enumerable` except that `Queryable` methods don't include `ToDictionary`, `ToDictionary` and `ToLookup`.

*The `System.Linq.Expressions` namespace contains generic LINQ expressions that can be used by any LINQ implementation. The `System.Linq.Expressions.ExpressionType` enumeration lists the 46 different types of expressions. The `System.Data.Linq.Expressions` namespace contains LINQ to SQL-specific expression types.*

### C# 3.0 Expression Trees

As an example, this conventional C# lambda expression compiles into IL that initializes a delegate to test a collection of integers for even values greater than 5:

```
Func<int, bool> isGt5andEven = x => x > 5 && x % 2 == 0;
```

## Part II: Introducing Language Integrated Query

---

Wrapping `Func<type, type, ...>` with `Expression<...>`, as in this example, causes the compiler to generate a tree of objects that *represent the expression*:

```
Expression<Func<int, bool>> isGt5andEven = x => x > 5 && x % 2 == 0;
```

The C# compiler expands the source code to this:

### C# 3.0

```
LambdaExpression isGt5andEven =
    Expression.Lambda(Expression.AndAlso(
        Expression.GreaterThan(param =
            Expression.Parameter(typeof(int), "x"),
        Expression.Constant(5)),
        Expression.Equal(
            Expression.Modulo(param,
                Expression.Constant(2)),
            Expression.Constant(0))),
    new ParameterExpression[] { param });
```

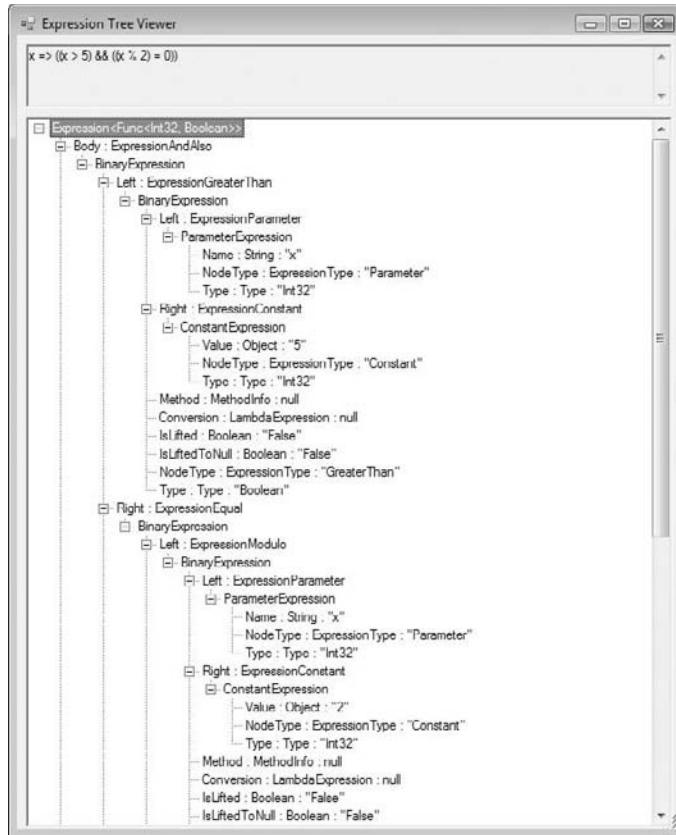
And the compiler finally generates IL for an expression tree. Figure 2-2 shows the `ExpressionTreeVisualizer`'s representation of the expanded C# code for the `isGt5andEven` lambda expression. This expression tree uses five of the 46 expression types: `Parameter`, `Constant`, `GreaterThanOrEqual`, `Modulo`, and `Lambda`.

## VS 2008 SP1's *ExpressionTreeVisualizer*

Expression trees aren't easy to understand, let alone write. To add your understanding of expression tree structure and syntax, the C# team created an `ExpressionTreeVisualizer`. The `ExpressionTreeVisualizer` ships as part of VS2008 SP1's Microsoft C# LINQ Samples and includes a simple Windows form application to launch the visualizer, but the Visualizer isn't installed by default. To install and run the visualizer with the preceding expression, do the following:

1. Navigate to your `\Program Files\Visual Studio 9.0\Samples\1033` folder, open `CSharpSamples.zip`, and extract the files to `\LinqSamples\CSharp` or the like with the Use Folder Names option selected.
2. Navigate to `\LinqSamples\CSharp\LinqSamples\ExpressionTreeVisualizer`, open `ExpressionTreeVisualizer.sln`, and press F5 to build and run the solution, which contains three projects: `ExpressionTreeVisualizer`, `ExpressionTreeVisualizerApplication`, and `ExpressionTreeGuiHost`.
3. Set `ExpressionTreeVisualizerApplication` as the StartUp Project.
4. Open `ExpressionTreeVisualizerApplication.Program.cs` and replace the default expression, `Expression<Func<int, bool>> expr = x => x == 10;` with `Expression<Func<int, bool>> isGt5andEven = x => x > 5 && x % 2 == 0;`.
5. Replace the `VisualizerDevelopmentHost func` argument name with `isGt5andEven`.
6. Press F5 to build and run the solution and view the resulting expression tree, part of which is shown in Figure 2-2.

Chapter 2: Understanding LINQ Architecture and Implementation



**Figure 2-2**

Figure 2-3 diagrams the process of “walking” the `isGt5andEven` expression tree.

## Part II: Introducing Language Integrated Query

---

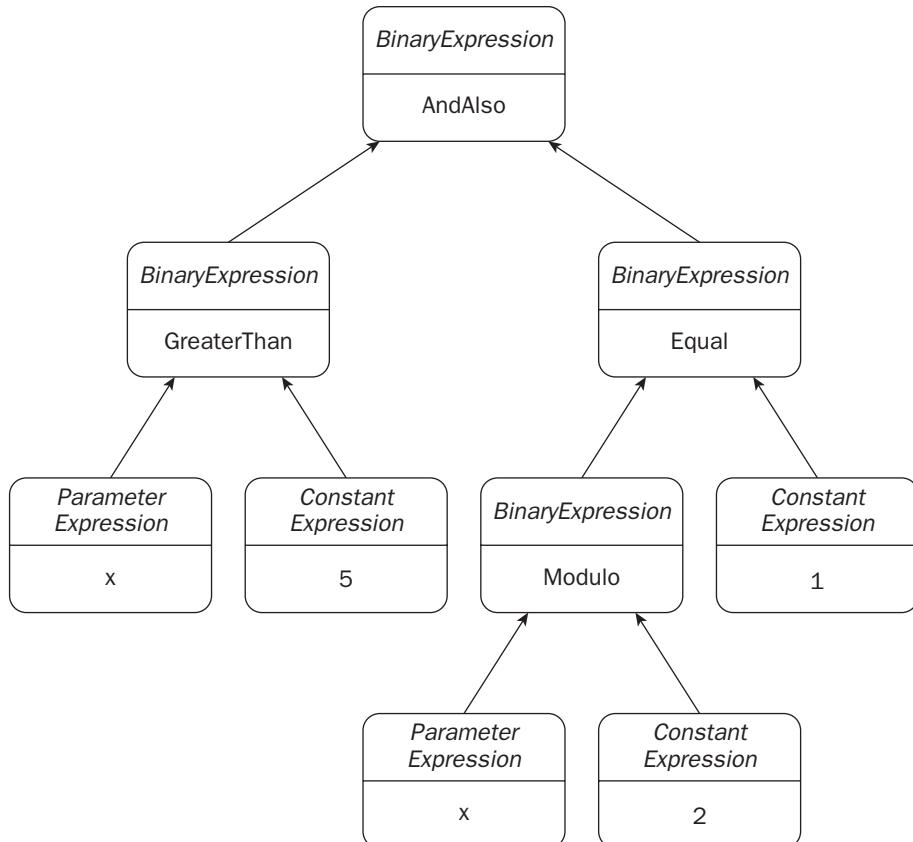


Figure 2-3

To install the ExpressionTreeVisualizer for debugging C# or VB applications, copy the `ExpressionTreeVisualizer.dll` assembly from the location in which you created it to Vista's \Users\UserName\Documents\Visual Studio 2008\Visualizers folder or the corresponding Windows XP folder. Optionally, copy it to the \Program Files\Microsoft Visual Studio 9.0\Common7\Packages\Debugger\Visualizers folder for use by all accounts.

## VB 9.0 Expression Trees

The syntax for VB 9.0 expression trees is similar to that for C# 3.0 versions but slightly more verbose. Here's the wrapped lambda expression method that corresponds to the preceding section's C# implementation:

```
Dim isGt5andEven As Expression(Of Func(Of Integer, Boolean)) = _  
Function(ByVal x As Integer) x > 5 AndAlso x Mod 2 = 1
```

## Chapter 2: Understanding LINQ Architecture and Implementation

---

The compiler generates the following source code with `expression5` substituting for `param` in the C# version:

```
Dim isGt5andEven As Expression(Of Func(Of Integer, Boolean)) = _
    Expression.Lambda(Of Func(Of Integer, Boolean)) _
    (Expression.AndAlso(Expression.GreaterThan(expression5 = _
        Expression.Parameter(GetType(Integer), "x"), _
        Expression.Constant(5, GetType(Integer))), True, Nothing), _
    Expression.Equal( _
        Expression.Modulo(expression5, _
        Expression.Constant(2, GetType(Integer))), _
        Expression.Constant(1, GetType(Integer))), True, Nothing)), _
New ParameterExpression() { expression5 })
```

### Compiled Queries

Generating complex T-SQL statements by interpreting the expression tree each time your application calls the `GetEnumerator()` method, such as in a `foreach/For Each` block, consumes substantial resources. If your query doesn't return many instances and the underlying instances don't contain property values that have a large number of bytes, compiling the query can result in performance improvement ratios in the range of 3:1 to 5:1.

Here's the C# and VB code for compiling a simple LINQ to SQL query against the Northwind Orders table:

#### C# 3.0

```
static class Queries
// Compiled query returns Orders instances for a specified country
{
    public static Func<NwindOrdersDataContext, string, IOrderedQueryable<Order>>
        OrdersByCountry = CompiledQuery.Compile((NwindOrdersDataContext nwdc,
            string country)
    => from o in nwdc.Orders
        where o.ShipCountry == country
        orderby o.OrderID descending
        select o);
}

private IEnumerable<Order> GetOrdersByCountry(string country)
{
    return Queries.OrdersByCountry(dcOrders, country);
}
```

#### VB 9.0

```
Friend NotInheritable Class Queries
    Public Shared OrdersByCountry As _
        Func(Of LINQtoSQLPerfVB.NorthwindDataContext, _
        String, IQueryable(Of Order)) = _
        CompiledQuery.Compile(Function(nwdc As LINQtoSQLPerfVB.NorthwindDataContext, _
            country As String) From o In nwdc.Orders _
```

## Part II: Introducing Language Integrated Query

---

```
    Where o.ShipCountry = country _
    Order By o.OrderID Descending _
    Select o)
End Class

Public Function GetOrdersByCountry(ByVal country As String) _
    As IEnumerable(Of Order)
    ' Delegate of OrdersByCountry
    Dim dc As New NorthwindDataContext()
    Return Queries.OrdersByCountry(dc, country)
End Function
```

The code in this section is taken from the \WROX\ADONET\Chapter05\LINQtoSQLPerfCS.sln and LINQtoSQLPerfVB.sln projects.

The static `OrdersByCountry()` function, which returns an `IOrderedQueryable<Order>` delegate type, and its `GetOrdersByCountry()` delegate function combine to provide a cached, compile-on-demand feature that's independent of `DataContext` instances. The next section discusses the `IQueryable<T>` and `IOrderedQueryable<T>` types.

This snippet calls the delegate function and iterates the compiled query expression:

### C# 3.0

```
IEnumerable<Order> compiledQuery =
    GetOrdersByCountry(strCountry).AsEnumerable<Order>();
foreach (var i in compiledQuery)
    lngTotal += i.OrderID;
```

### VB 9.0

```
Dim Query = GetOrdersByCountry(strCountry).AsEnumerable()
For Each i In Query
    lngTotal += i.OrderID
Next i
```

Chapter 5 digs deeper into the LINQ Expression Tree API and demonstrates the performance improvements you can expect from compiling your LINQ to SQL queries.

## The `IQueryable<T>` Interface and Domain-Specific LINQ Implementations

The `IQueryable<T>` and `IOrderedQueryable<T>` types are *not* collections; they are descriptions of a LINQ query that supports polymorphism, optimization, and dynamic query generation. However, you can invoke `IQueryable<T>`'s `ToList()` method to call `GetEnumerator()` and convert the sequence to a `List<T>` type.

*The `IOrderedQueryable<T>` type is required to support query expressions that include `orderby/Order By` only, so the remainder of this section uses `IQueryable<T>` to represent both implementations, as well as the nongeneric `IQueryable` and `IOrderedQueryable` types.*

## Chapter 2: Understanding LINQ Architecture and Implementation

---

During LINQ's early development stage, third-party developers and independent software vendors (ISVs) were forced to define their own set of query operators to implement the familiar `from...in...` `where...` `orderby...` `select` structure and other SQOs for extending LINQ to other data domains. Doing this was such a daunting task that no rational third-party would even consider devoting the necessary resources to a new and then-unproven technology like LINQ.

Providing an `AsQueryable()` method for `IEnumerable<T>` enables SQOs applied to `IQuerables` to be captured as expression trees and compiled to IL with a compiler add-in. Writing the code to implement and interpret or compile expression trees is a snap compared with that for reimplementing the entire set of SQOs. Providing an `AsEnumerable()` method for `IQueryables` enables local query processing for in-memory objects. The LINQ to SQL team convinced the LINQ to Objects group to move the expression tree compiler from the LINQ to SQL implementation to the core LINQ product, which enabled the use of the standard SQOs. Moving the required Extension types from the `System.Data.Linq` to the `System.Linq` namespace makes them available to Silverlight, the next version of VB, JavaScript, and the new dynamic .NET languages — IronPython and IronRuby.

`IQueryable<T>` is now LINQ's focal point for extensions by domain-specific query processor add-ins and dynamic queries. As witness to the success of `IQueryable<T>` and expression tree approach, there were more than 10 third-party LINQ implementations in development by individual programmers before VS 2008 released to manufacturing. Chapter 8 covers several useful examples of LINQ extensions by early adopters.

## Summary

This chapter provided a fast-paced introduction to the new namespaces of the .NET Framework 3.5 and new keywords that VS 2008 introduces into VB 9.0 and C# 3.0 to support LINQ. The chapter also covered the following features of the VB 9.0 and C# 3.0 compilers than enable LINQ for Objects and domain-specific LINQ implementations:

- ❑ Implicitly typed local variables
- ❑ Object initializers
- ❑ Array initializers\*
- ❑ Collection initializers\*
- ❑ Anonymous types
- ❑ Extension methods
- ❑ Lambda expressions
- ❑ Standard Query Operators
- ❑ Query expressions
- ❑ Expression trees
- ❑ `IQueryable<T>` and `IOrderedQueryable<T>`

\* VB 9.0 doesn't support these features due to insufficient time, resources, or both devoted by the VS 2008 team to Visual Basic.



# 3

## Executing LINQ Query Expressions with LINQ to Objects

LINQ to Objects is Visual Studio 2008's core LINQ implementation for querying in-memory objects. All other Microsoft and third-party LINQ implementations use the same or a subset of the LINQ to Objects query expression syntax because the query keywords are part of the C# 3.0 and VB 9.0 languages. The two language compilers translate LINQ to Objects query expressions into the method call syntax, which chains keyword extension methods to an `IEnumerable<T>` type. The extension methods use lambda expressions as parameters. Chapter 2's "Standard Query Operators" section gave you an overview of the translation to method call process. The compilers translate the lambda expressions to anonymous methods and use anonymous type syntax to auto-generate private classes that compile to .NET Intermediate Language (MSIL). The syntax and code examples in this and the next chapter operate on `IEnumerable<T>` and `IOrderedEnumerable<T>` types that the compiler processes by this rather roundabout approach.

*Some writers refer to LINQ's method call syntax as explicit syntax. However, calling methods to which you assign lambda expressions as predicates is no more explicit than query expressions based on keywords; both query dialects compile to the same MSIL. In fact, keyword-based query expressions probably are more explicit than method calls because they're easier for human beings to read and interpret.*

Microsoft's LINQ to SQL, LINQ to DataSets, LINQ to Entities and LINQ to XML, as well as third-party LINQ implementations for domain-specific query languages (DSQLs), don't translate and compile query expressions to IL directly. Instead, the compiler generates at runtime an in-memory representation of the query as an expression tree, which contains custom code to translate the LINQ query expressions to the DSQL required by the implementation. As examples, LINQ to SQL generates Transact-SQL (T-SQL), LINQ to Entities generates Entity SQL (eSQL), and LINQ to XML generates XPath statements. The output of these queries is an `IQueryable<T>` object rather than an `IEnumerable<T>` type. (`IQueryable<T>` implements the `IEnumerable<T>` interface also). Part III covers these implementations (except LINQ to Entities), as well as those by third parties, and

## Part II: Introducing Language Integrated Query

---

their expression trees. Chapter 12 is devoted to the LINQ implementation for the ADO.NET Entity Framework.

The fact that the keywords used to compose LINQ query expressions are part of the C# 3.0 and VB 9.0 language specifications means that all LINQ implementations for `IEnumerable<T>` and `IQueryable<T>` types must deal with the set or a subset of LINQ's Standard Query Operators (SQOs). Therefore, it's important that you become familiar with the available SQOs before advancing to domain-specific implementations in Part III and LINQ to Entities in part IV.

*Microsoft's "The .NET Standard Query Operators" whitepaper is available on the Web at <http://msdn2.microsoft.com/en-us/library/bb394939.aspx>. A February 2007 update was the latest version available when this book was written.*

## Standard Query Operators by Group

The 50 SOQs are classified into 14 groups. C# 3.0 and VB 9.0 projects use the same SOQs from the `System.Linq` namespace's `Enumerable` and `Queryable` classes. The `Enumerable` class provides a set of static methods for querying *objects* that implement the `IEnumerable<T>` interface. The `Queryable` class supplies a set of static methods for querying *data structures* that implement the `IQueryable<T>` interface.

The following table lists the groups and their operators in the same sequence as the "The .NET Standard Query Operators" technical specification of February 2007.

Operator Group	Operator	Operator Group	Operator
Restriction operator	Where	Element operators	First
Projection operators	Select		FirstOrDefault
	SelectMany		Last
Partitioning operators	Take		LastOrDefault
	Skip		Single
	TakeWhile		SingleOrDefault
	SkipWhile		ElementAt
Join operators	Join		ElementAtOrDefault
	GroupJoin		DefaultIfEmpty
Concatenation operator	Concat	Generation operators	Range

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

Operator Group	Operator	Operator Group	Operator
Ordering operators	OrderBy OrderByDescending ThenBy ThenByDescending Reverse	Quantifiers	Repeat Empty Any All Contains
Grouping operator	GroupBy	Aggregate operators	Count
Set operators	Distinct Union Intersect Except		LongCount Sum Min Max
Conversion operators	AsEnumerable AsQueryable ToArray ToList ToDictionary ToLookup OfType Cast	Unclassified keywords	Average Aggregate let/Let equals/Equals from/From in/In into/Into Key
Equality operator	SequenceEqual		

\* The unclassified keywords (with the exception of `let/Let`) appear in the May 2006 edition of “The .NET Standard Query Operators” technical specification but aren’t classified by type.

This chapter presents descriptions of and sample C#/VB queries from the SQOs in the table’s sequence, except for the unclassified keywords that are used in the applicable query expression context.

## SQOs as Keywords in C# 3.0 and VB 9.0

LINQ-specific query keywords are, for the most part, a very small subset of ANSI SQL-92 reserved words. The LINQ development team chose SQL — rather than XPath or XQuery — as the primary model for LINQ’s query expression language because most .NET developers are at least familiar with SQL. The most obvious difference between SQL and LINQ syntax is the inversion of the sequence of the `select/Select` and `from/From` clauses and the requirement for an alias variable to reference each source object.

## Part II: Introducing Language Integrated Query

---

Microsoft's "C# Language Specification Version 3.0" defines a query expression as follows: "A query expression begins with a from clause and ends with either a select or group clause." The Visual Basic definition differs only in the case of the from, select and group clause names. Both languages recognize query expression keywords only within query expressions that begin with from or From.

The following table lists SQOs and unclassified operators that are keywords in C# 3.0 from ... select and VB 9.0 From ... Select expressions:

C# 3.0	VB 9.0	C# 3.0	VB 9.0
	Aggregate	join	Join
	Distinct	Key	Key
equals	Equals	let	Let
from	From	orderby	Order By
group [alias] by	Group By	select	Select
	Group Join		Skip
in	In		Take
into	Into	where	Where

Keywords in the preceding table correspond to SQOs except equals, Equals, in, In, into, Into, Key, let, and Let. The in and In prepositions are used only in from/From clauses. You use the C# equals and into keywords with the join; the compiler translates join ... into clauses into GroupJoin method calls. VB's Group Join keyphrase enables adding the Into keyword and translates Group Join ... Into statements into GroupJoin method calls. VB uses Group as a range variable name in Group Join ... Into Group and Group By ... Into Group clauses. Several keywords and operators use the Key operator, which determines group membership in query expressions that use the group alias by and Group Alias By expressions.

All SQOs may be chained in a query expressed with method call syntax. Those SQOs that don't correspond to keywords of a specific language must be chained to a query expression that is enclosed in parenthesis. VB has five keywords that C# doesn't support: Aggregate, Distinct, Group, Skip and Take. The following two query expressions demonstrate differing syntax for the Skip and Take operators in C# and VB:

### C# 3.0

```
(select c from customers)
.Skip(5)
.Take(5)
```

### VB 9.0

```
Select c From Customers _
Skip 5 _
Take 5
```

Paging iterator output is the primary application for the Skip() and Take() operators.

## The LINQ Project Sample Query Explorers

VS 2008 Standard Edition and higher Setup program installs LINQ and other sample files as CSharpSamples.zip and VBSamples.zip archives. The \Program Files\Microsoft Visual Studio 2008\Samples\1033 folder for the U.S. English (ENU) localization contains the two .zip files. Here's how to expand the archives and locate the C# 3.0 and VB 9.0 LINQ Project Sample Query Explorer projects:

- ❑ Expand ... \CSharpSamples.zip with the Use Folder Names check box marked (required) into a folder of your choice to create ... \Language Samples and ... \LINQ Samples subfolders. The C# LINQ Sample Query Explorer project (QuerySamples.sln) is in the ... \LINQ Samples \QuerySamples folder.
- ❑ Expand ... \VBSamples.zip with the Use Folder Names check box marked into a folder of your choice to create a ... \VB Samples subfolder with several subfolders. The VB LINQ Sample Query Explorer project (QuerySamples.sln) is in the ... \Language Samples\LINQ Samples\QuerySamples folder.

Running QuerySamples.sln's C# 3.0 and VB 9.0 versions let you select and execute sets of LINQ to Objects, LINQ to SQL, LINQ to XML, LINQ to DataSet, and LINQ to Entities queries.

Each set has 101 or more sample queries classified into groups similar to those for the SQOs. Most examples use query expressions but some demonstrate method call syntax. Figure 3-1 shows the C# Query Explorer with the 101 LINQ Query Samples (LINQ to Objects) set selected, the Restriction Operators and Projection Operators nodes expanded, and the Where – Drilldown sample executed.

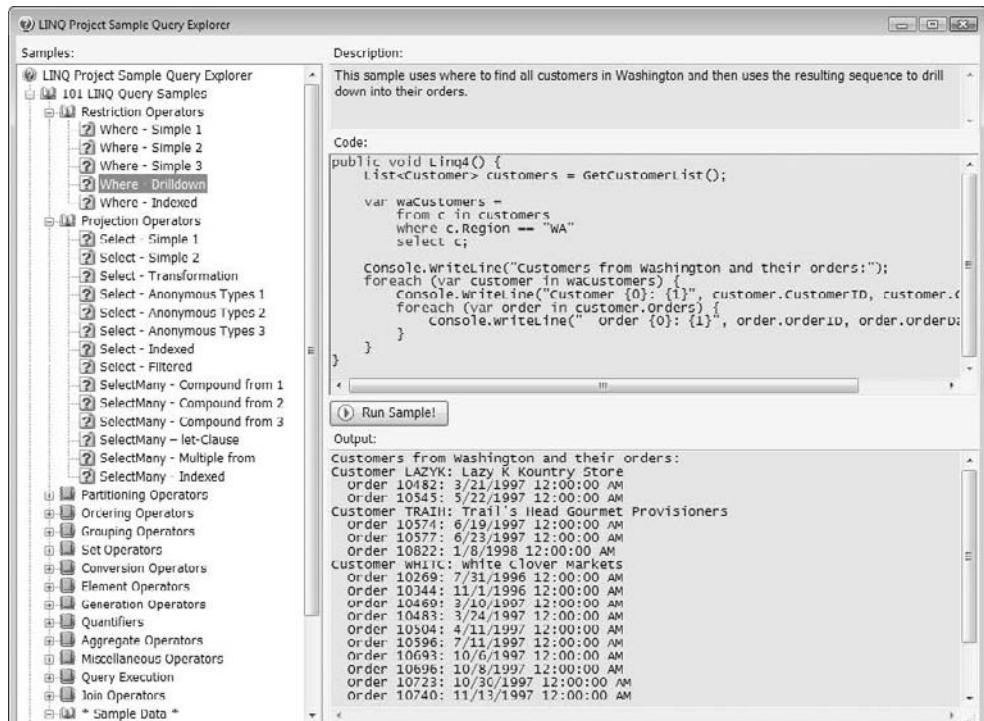


Figure 3-1

## Part II: Introducing Language Integrated Query

---

The LINQ to Objects examples use integer and string arrays to demonstrate simple query expressions. In-memory `productList` and `customerList` instances serve as source data for more complex queries. Customer instances in `customerList` include an `Orders` field of a `List<Order>` collection holding objects associated with individual Customer objects. `ObjectDumper`, which Chapter 1's "Applying Operators to Emulate SQL Functions, Modifiers, and Operators" section introduced, redirects Console output to the Query Explorer's text box.

*Microsoft's 101 LINQ Samples page (<http://msdn2.microsoft.com/en-us/vcsharp/aa336746.aspx>) has links to the C# sample queries, which are organized by group. Clicking a link displays the query expression and its result. This page doesn't contain the Join Operators group.*

## Sample Classes for LINQ to Objects Code Examples

The source code for the two sample Query Explorer projects is too complex to make multiple ad-hoc LINQ query edits and builds practical. So this chapter uses several sample C# and VB WinForm projects that generate multiple, related in-memory objects to serve as source data. A LINQ in-memory object generator (LIMOG) utility program writes the C# 3.0 or VB 9.0 class declarations for representative business objects, which are more complex than those used by the LINQ Project Sample Query Explorers. Any database in an accessible SQL Server [Express] Instance can provide the source data and metadata for LIMOG. LIMOG is optimized for the naming conventions of the Northwind and OakLeafU sample databases, but should be adaptable to 1:1 table:class mapping for moderately complex databases.

LIMOG lets you incorporate bidirectional (1:1 and 1:many associations between related objects to create business object graphs. It then writes C# 3.0 or VB 9.0 instance initialization code for `List<T>` collections from the database, tables, columns, and foreign and primary keys that you specify. Figure 3-2 illustrates LIMOG in the process of generating the C# Order class definition and initialization code for the `List<Order>` collection that's described in the next section.

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

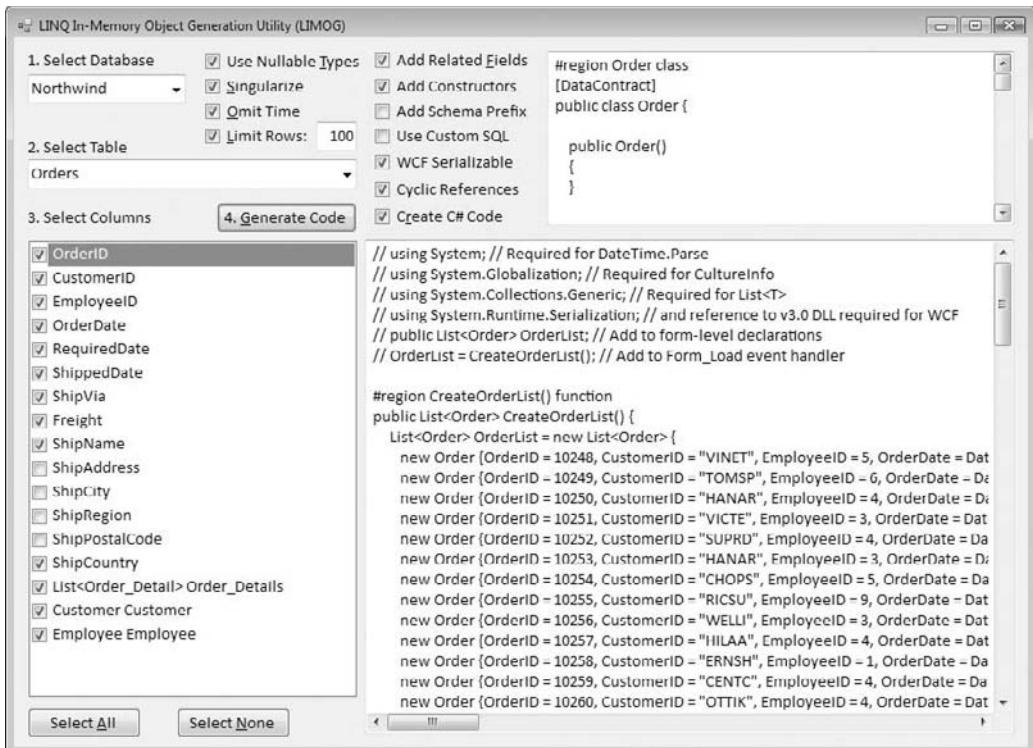


Figure 3-2

A second, database-specific pair of test harnesses generate the `List<Object>` collections, populate the object references and lists for associations between related objects, and persist (*serialize* or *dehydrate*) the complete object graph in C# and VB binary files. The graph consists of plain old CLR objects (POCOs) with public fields instead of properties with getters and setters for simplicity. (C# lets you add automatic properties or refactor fields to full-fledged properties with the new Field Encapsulate feature.) The `[Serializable]` or `<Serializable()>` attribute, which is required by the .NET binary serialization process, is the only .NET-specific attribute applied to the classes. Chapter 4 describes the LIMOG.sln project and the use of LINQ GroupJoin queries to populate associations.

*You'll find the LIMOG.sln VB 9.0 project in the \WROX\ADONET\Chapter 04 folder. The \WROX\ADONET\Chapter04\C#\NwindObjectsCS.sln and \WROX\ADONET\Chapter04\VB#\NwindObjectsVB.sln projects generate the 450-KB NwindGraphCS.dat and NwindGraphVB.dat files in the projects' ... \bin\debug folders. This chapter's ... \bin\debug folders contain copies of these files.*

This chapter's sample projects load the NwindGraphCS.dat or NwindGraphVB.dat file on loading, which requires less than 300 ms with a fast computer.

Figure 3-3 is a Visio-style entity-relationship diagram of the eight classes available for use by this chapter's sample LINQ to Objects queries.

## Part II: Introducing Language Integrated Query

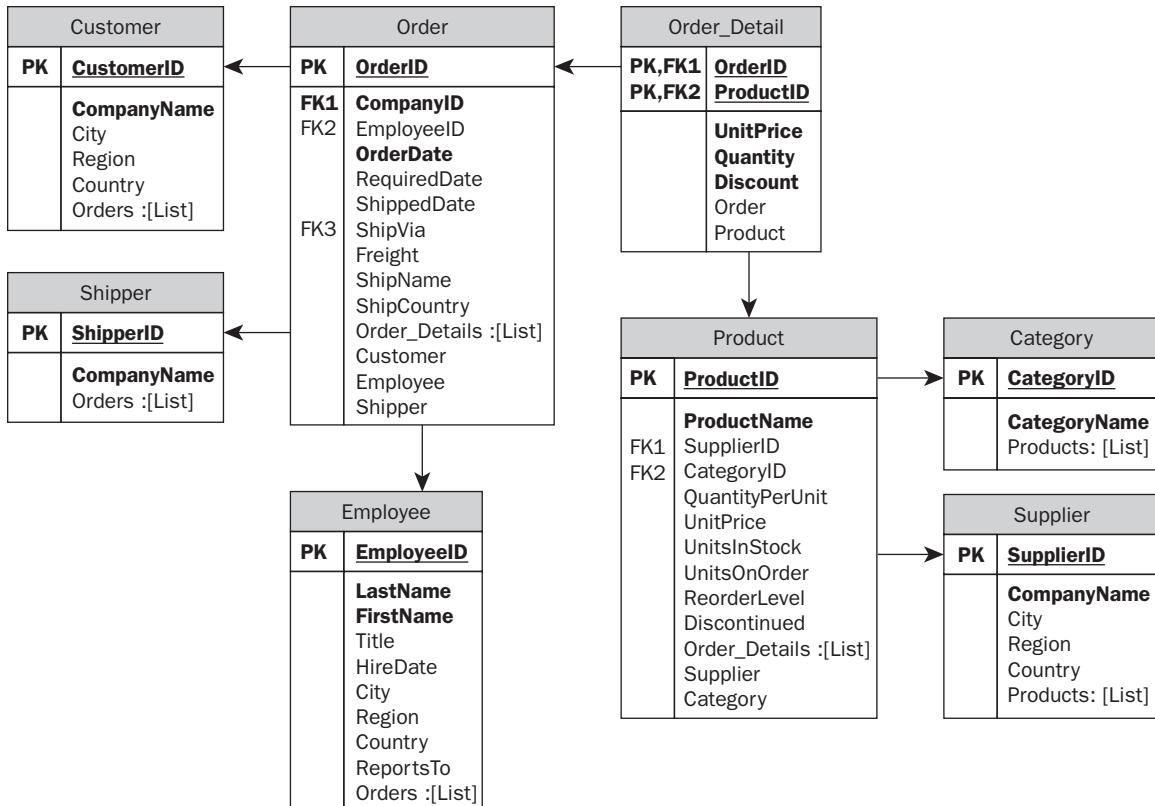


Figure 3-3

The `ClassName : [List]` fields below the last conventional field in Figure 3-3 represent 1:many associations with related objects. Other fields named for classes represent 1:1 associations with the named class. These fields are emphasized in the following four code examples.

### C# Class Definition and Initialization Code Example

The selections shown in Figure 3-2 generate the following C# 3.0 class definition file:

```
public class Order {
    public int OrderID;
    public string CustomerID;
    public int? EmployeeID;
    public DateTime? OrderDate;
    public DateTime? RequiredDate;
    public DateTime? ShippedDate;
    public int? ShipVia;
    public decimal? Freight;
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

```
    public string ShipName;
    public string ShipCity;
    public string ShipRegion;
    public string ShipCountry;
    public List<Order_Detail> Order_Details;
    public Customer Customer;
    public Employee Employee;
    public Shipper Shipper;
}
```

Omitting fields, such as shipping addresses, that aren't likely to be used as `Where` clause restrictions, `Join` or `GroupJoin` fields, or `GroupBy` predicates speeds .dat file loading and deserialization (*object hydration*) on project startup.

Following is the `CreateOrderList()` function's abbreviated C# collection initializer code to populate the C# `OrderList` collection:

### C# 3.0

```
public List<Order> CreateOrderList() {
    List<Order> OrderList = new List<Order> {
        new Order {OrderID = 10248, CustomerID = "VINET", EmployeeID = 5,
            OrderDate = DateTime.Parse("7/4/1996",
                CultureInfo.CreateSpecificCulture("en-US"))),
            RequiredDate = DateTime.Parse("8/1/1996",
                CultureInfo.CreateSpecificCulture("en-US")),
            ShippedDate = DateTime.Parse("7/16/1996",
                CultureInfo.CreateSpecificCulture("en-US"))),
            ShipVia = 3, Freight = 32.3800M, ShipName = "Vins et alcools Chevalier",
            ShipCountry = "France", Order_Details = new List<Order_Detail>(),
            Customer = new Customer(), Employee = new Employee(),
            Shipper = new Shipper()),
        new Order {OrderID = 10249, CustomerID = "TOMSP", EmployeeID = 6,
            OrderDate = DateTime.Parse("7/5/1996",
                CultureInfo.CreateSpecificCulture("en-US"))),
            RequiredDate = DateTime.Parse("8/16/1996",
                CultureInfo.CreateSpecificCulture("en-US")),
            ShippedDate = DateTime.Parse("7/10/1996",
                CultureInfo.CreateSpecificCulture("en-US"))),
            ShipVia = 1, Freight = 11.6100M, ShipName = "Toms Spezialitäten",
            ShipCountry = "Germany", Order_Details = new List<Order_Detail>(),
            Customer = new Customer(), Employee = new Employee(),
            Shipper = new Shipper()),
        new Order {OrderID = 10347, CustomerID = "FAMIA", EmployeeID = 4,
            OrderDate = DateTime.Parse("11/6/1996",
                CultureInfo.CreateSpecificCulture("en-US"))),
            RequiredDate = DateTime.Parse("12/4/1996",
                CultureInfo.CreateSpecificCulture("en-US")),
            ShippedDate = DateTime.Parse("11/8/1996",
```

## Part II: Introducing Language Integrated Query

```
        CultureInfo.CreateSpecificCulture("en-US"))), ShipVia = 3,
        Freight = 3.1000M, ShipName = "Familia Arquibaldo",
        ShipCountry = "Brazil", Order_Details = new List<Order_Detail>(),
        Customer = new Customer(), Employee = new Employee(),
        Shipper = new Shipper())
    };
    return OrderList;
}
```

C# requires parsing string `DateTime` values to return a valid `Date` object without time values. The code specifies `CultureInfo.CreateSpecificCulture("en-US")` to accommodate non-US locales if the `OmitTime` option is selected.

## VB Class Definition and Initialization Code Example

Following is the VB 9.0 class definition for the `Order` object:

```
Public Class Order
    Public OrderID As Int32
    Public CustomerID As String
    Public EmployeeID As Int32?
    Public OrderDate As DateTime?
    Public RequiredDate As DateTime?
    Public ShippedDate As DateTime?
    Public ShipVia As Int32?
    Public Freight As Decimal?
    Public ShipName As String
    Public ShipCity As String
    Public ShipRegion As String
    Public ShipCountry As String
    Public Order_Details As List(Of Order_Detail)
    Public Customer As Customer
    Public Employee As Employee
    Public Shipper As Shipper
End Class
```

VB 9.0 doesn't offer collection initializer syntax, but it's easy to add individual object initializers, as shown here:

```
Public Function CreateOrderList() As List(Of Order)
    OrderList = New List(Of Order)
    With OrderList
        .Add(New Order With {.OrderID = 10248, .CustomerID = "VINET",
            .EmployeeID = 5, .OrderDate = #7/4/1996#, .RequiredDate = #8/1/1996#,
            .ShippedDate = #7/16/1996#, .ShipVia = 3, .Freight = 32.3800,
            .ShipName = "Vins et alcools Chevalier", .ShipCity = "Reims",
            .ShipCountry = "France", .Order_Details = New List(Of Order_Detail)(),
            .Customer = New Customer(), .Employee = New Employee(),
            .Shipper = New Shipper()}))
```

```
.Add(New Order With {.OrderID = 10249, .CustomerID = "TOMSP",
    .EmployeeID = 6, .OrderDate = #7/5/1996#, .RequiredDate = #8/16/1996#,
    .ShippedDate = #7/10/1996#, .ShipVia = 1, .Freight = 11.6100,
    .ShipName = "Toms Spezialitäten", .ShipCity = "Münster",
    .ShipCountry = "Germany", .Order_Details = New List(Of Order_Detail)(),
    .Customer = New Customer(), .Employee = New Employee(),
    .Shipper = New Shipper()})
...
.Add(New Order With {.OrderID = 10347, .CustomerID = "FAMIA",
    .EmployeeID = 4, .OrderDate = #11/6/1996#, .RequiredDate = #12/4/1996#,
    .ShippedDate = #11/8/1996#, .ShipVia = 3, .Freight = 3.1000,
    .ShipName = "Familia Arquibaldo", .ShipCity = "Sao Paulo",
    .ShipRegion = "SP", .ShipCountry = "Brazil",
    .Order_Details = New List(Of Order_Detail)(),
    .Customer = New Customer(), .Employee = New Employee(),
    .Shipper = New Shipper()})
End With
Return OrderList
End Function
```

Enclosing DateTime strings with the # symbol eliminates the need to apply the `DateTime.Parse()` method to the string. The VB samples assume that the user's culture is en-US. The NwindObjectsCS.sln and NwindObjectVB.sln projects add 10 years to the Orders table's original twentieth century dates.

## Restriction Operator: Where

The `Where` restriction operator filters source sequences according to its predicate expression in a manner similar to the SQL `WHERE` clause with the same predicate. Here are the common signatures for the `Where` method:

### C# 3.0

```
public static IEnumerable<TSource> Where<TSource> (
    IEnumerable<T> source,
    Func<TSource, bool> predicate
)
```

### VB 9.0

```
Public Shared Function Where(Of TSource) ( _
    Source As IEnumerable(Of TSource), _
    Predicate As Func(Of TSource, Boolean) _
) As IEnumerable(Of TSource)
```

*You'll find the sample code for this chapter as QueryOperatorsCS.sln and QueryOperatorsVB.sln in the \WROX\ADONET\Chapter03\C#\QueryOperators and \WROX\ADONET\Chapter03\VB\QueryOperators folders, respectively.*

## Part II: Introducing Language Integrated Query

---

### Simple Where Expressions

Following are simple `Where` operator usage examples of method call and query expression syntax to filter `Customer` instances for the USA and display three fields with an iterator:

#### C# 3.0

```
var query1 = CustomerList.Where(c => c.Country == "USA");
query1 = from c in CustomerList where c.Country == "USA" select c;

foreach (var c in query1)
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}, " +
        "Country = {2}\r\n", c.CustomerID, c.CompanyName, c.Country));
```

The method call expects the expression (predicate) to be passed as a lambda function. Notice that C# query expressions require a terminating `Select` instruction; method calls don't need termination in either language. VB emulates `Select AliasName` if it's missing, as demonstrated here:

#### VB 9.0

```
Dim Query1 = CustomerList.Where(Function(c) c.Country = "USA")
Query1 = From c In CustomerList Where c.Country = "USA"

For Each c In Query1
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}, " & _
        "Country = {2}" & vbCrLf, c.CustomerID, c.CompanyName, c.Country))
Next
```

The compiler translates the `where/Where` clauses the `Where()` method call example above the sample query expression.

Both snippets produce the following output (abbreviated from 13 rows with an unmodified Customers table):

```
CustomerID = GREAL, CompanyName = Great Lakes Food Market, Country = USA
CustomerID = HUNGC, CompanyName = Hungry Coyote Import Store, Country = USA
...
CustomerID = WHITC, CompanyName = White Clover Markets, Country = USA
```

### Compound Where Expressions

The following snippets demonstrate a compound `Where` expression with the `&&` and `And` operators, which use the iterator to take advantage of the associations between `Customer` and `Order` and `Order_Detail` instances. The resulting hierarchical list doesn't require a `Select` or `SelectMany` projection:

#### C# 3.0

```
var query2 = CustomerList.Where(c => c.Country == "USA" && c.Orders.Any());
query2 = from c in CustomerList where c.Country == "USA" && c.Orders.Any()
select c;

foreach (var c in query2)
{
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

```
sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}, " +
    "Country = {2}\r\n", c.CustomerID, c.CompanyName, c.Country));
foreach (var o in c.Orders)
{
    sbResult.Append(String.Format("    OrderID = {0}, EmployeeID = {1}, " +
        "OrderDate = {2:d}\r\n", o.OrderID, o.EmployeeID, o.OrderDate));
    foreach (var d in o.Order_Details)
        sbResult.Append(String.Format("        OrderID = {0}, " +
            "ProductID = {1}, UnitPrice = {2:c}\r\n", d.OrderID,
            d.ProductID, d.UnitPrice)));
}
}
```

The Any() quantifier returns true if the `IEnumerable<T>` type that invokes it contains any elements; the Any quantifier is equivalent to the `Count() > 0` expression. There is no limit to the number of expressions that return `bool` or Boolean values that you can combine with `&&` and `||` or `And`, `and` or `Or` operators.

### VB 9.0

```
Dim Query2 = CustomerList.Where(Function(c) c.Country = "USA" And c.Orders.Any())
Query2 = From c In CustomerList Where c.Country = "USA" And c.Orders.Any()

For Each c In Query2
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}, " & _
        "Country = {2}" & vbCrLf, c.CustomerID, c.CompanyName, c.Country))
    For Each o In c.Orders
        sbResult.Append(String.Format("    OrderID = {0}, EmployeeID = {1}, " & _
            "OrderDate = {2:d}" & vbCrLf, o.OrderID, o.EmployeeID, o.OrderDate))
        For Each d In o.Order_Details
            sbResult.Append(String.Format("        OrderID = {0}, " & _
                "ProductID = {1}, UnitPrice = {2:c}" & vbCrLf, d.OrderID,
                d.ProductID, d.UnitPrice)))
        Next
    Next
Next
```

The resulting hierarchical list contains no rows for `CustomerID = GREAL, HUNGC, LAZYKJ, LETTS, THECR, TRAIH` because these customers placed no orders during the interval represented by the sample's 100 Order instances (7/4/2006 to 11/6/2006). Here's the abbreviated result:

```
CustomerID = LONEP, CompanyName = Lonesome Pine Restaurant, Country = USA
    OrderID = 10307, EmployeeID = 2, OrderDate = 9/17/2006
        OrderID = 10307, ProductID = 62, UnitPrice = $39.40
        OrderID = 10307, ProductID = 68, UnitPrice = $10.00
    OrderID = 10317, EmployeeID = 6, OrderDate = 9/30/2006
        OrderID = 10317, ProductID = 1, UnitPrice = $14.40
CustomerID = OLDWO, CompanyName = Old World Delicatessen, Country = USA
    OrderID = 10305, EmployeeID = 8, OrderDate = 9/13/2006
        OrderID = 10305, ProductID = 18, UnitPrice = $50.00
        OrderID = 10305, ProductID = 29, UnitPrice = $99.00
        OrderID = 10305, ProductID = 39, UnitPrice = $14.40
```

## Part II: Introducing Language Integrated Query

---

```
OrderID = 10338, EmployeeID = 4, OrderDate = 10/25/2006
OrderID = 10338, ProductID = 17, UnitPrice = $31.20
OrderID = 10338, ProductID = 30, UnitPrice = $20.70
...
CustomerID = WHITC, CompanyName = White Clover Markets, Country = USA
OrderID = 10269, EmployeeID = 5, OrderDate = 7/31/2006
OrderID = 10269, ProductID = 33, UnitPrice = $2.00
OrderID = 10269, ProductID = 72, UnitPrice = $27.80
OrderID = 10344, EmployeeID = 4, OrderDate = 11/1/2006
OrderID = 10344, ProductID = 4, UnitPrice = $17.60
OrderID = 10344, ProductID = 8, UnitPrice = $32.00
```

## **Method Calls with Index Arguments and Use of IndexOf()**

SQOs that return `IEnumerable<T>` collections have an overload that includes the index of the member. Including the index allows the member's position in the collection to be used in expressions. Index values correspond to those of the T-SQL `ROW_NUMBER()` function minus 1.

Index arguments are highlighted in the following method signatures and sample code:

### **C# 3.0**

```
public static IEnumerable<TSource> Where<TSource> (
    IEnumerable<TSource> source,
    Func<TSource, int, bool> predicate
)
```

### **VB 9.0**

```
Public Shared Function Where(Of TSource) ( _
    Source As IEnumerable(Of TSource), _
    Predicate As Func(Of TSource, Integer, Boolean)) _
) As IEnumerable(Of TSource)
```

You can use the index argument in method call comparison expressions that return a `bool/Boolean` value. These two expressions return the last five of the 13 elements iterated by the query of the earlier “Simple Where Expressions” section:

### **C# 3.0**

```
var query3 = CustomerList.Where((c, index) => c.Country == "USA" && index > 70);

foreach (var c in query3)
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}, " +
        "Country = {2}\r\n", c.CustomerID, c.CompanyName, c.Country));
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

### VB 9.0

```
Dim Query3 = CustomerList.Where(Function(c, index) c.Country = "USA" _  
    And index > 70)  
  
For Each c In Query3  
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}, " & _  
        "Country = {2}" & vbCrLf, c.CustomerID, c.CompanyName, c.Country))  
Next
```

*Query expressions don't support use of the index overload.*

Here's the output:

```
CustomerID = SPLIR, CompanyName = Split Rail Beer & Ale, Country = USA  
CustomerID = THEBI, CompanyName = The Big Cheese, Country = USA  
CustomerID = THECR, CompanyName = The Cracker Box, Country = USA  
CustomerID = TRAIH, CompanyName = Trail's Head Gourmet Provisioners, Country = USA  
CustomerID = WHITC, CompanyName = White Clover Markets, Country = USA
```

The 0-based index of the SPLIR record is 74 (of 91 CustomerList records).

You can achieve the same result by applying the `IndexOf(member)` method to the source collection, as in these examples, which include query expressions:

### C# 3.0

```
var query3 = CustomerList.Where(c => c.Country == "USA" &&  
    CustomerList.IndexOf(c) > 70);  
query3 = from cust in CustomerList  
    where cust.Country == "USA" && CustomerList.IndexOf(cust) > 70  
    select cust;
```

### VB 9.0

```
Dim Query3 = CustomerList.Where(Function(c, Index) c.Country = "USA" _  
    And CustomerList.IndexOf(c) > 70)  
Query3 = From c In CustomerList _  
    Where c.Country = "USA" And CustomerList.IndexOf(c) > 70
```

The index overload is similar (an `int` type added as `Func`'s second argument) for most SQO's that return `IEnumerable<T>` sequences. `IndexOf(member)` works with method call syntax and query expressions, so the remaining SQO descriptions (except the next) don't include the overload details.

## Projection Operators

Projection operators correspond to SQL's `SELECT ColumnList` clause. If you omit the `Select` or `SelectMany` operators from a method call or the `Select` clause from VB query expression, the `IEnumerable<T>` query variable contains hierarchical lists for the base as well as associated collections. Projection operators move control of the specific object properties to be displayed or processed from the iterator to the query. Of course, the iterator can further restrict the property set.

## Part II: Introducing Language Integrated Query

---

### Select

The `Select` operator returns from the source object a sequence with the set of properties specified by the selector that's passed to the lambda function. Here are the `Select` method signatures:

#### C# 3.0

```
public static IEnumerable<TResult> Select<TSource, TResult> (
    IEnumerable<TSource> source,
    Func<TSource, TResult> selector
)
```

#### VB 9.0

```
Public Shared Function Select(Of TSource, TResult) ( _
    source As IEnumerable(Of TSource), _
    selector As Func(Of TSource, TResult) _
) As IEnumerable(Of TResult)
```

### Simple Select Projection Expression

The following two snippets deliver 13 rows of text output for the first example of this chapter with each row enclosed by French braces:

#### C# 3.0

```
var query4 = CustomerList.Where(c => c.Country == "USA"
                               .Select(c => new { c.CustomerID, c.CompanyName,
                               c.Country });
query4 = from c in CustomerList
        where c.Country == "USA"
        select new {c.CustomerID, c.CompanyName, c.Country};

foreach (var c in query4)
    sbResult.Append(c.ToString() + "\r\n");
```

#### VB 9.0

```
Dim Query4 = CustomerList.Where(Function(c) c.Country = "USA") _
                           .Select(Function(c) New With {c.CustomerID, _
                           c.CompanyName, c.Country})
Query4 = From c In CustomerList _
          Where c.Country = "USA" _
          Select New With {c.CustomerID, c.CompanyName, c.Country}

For Each c In Query4
    sbResult.Append(c.ToString() & vbCrLf)
Next
```

These two examples move the column list from the iterator to the query body and enable the `ToString()` method to expand the resulting sequence's property name = value pairs, as shown here (abbreviated):

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

```
{CustomerID = GREAL, CompanyName = Great Lakes Food Market, Country = USA}  
{CustomerID = HUNGC, CompanyName = Hungry Coyote Import Store, Country = USA}  
...  
{CustomerID = WHITC, CompanyName = White Clover Markets, Country = USA}
```

### Multiple Select Projection Expression with Index Value

The index value is a property of the sequence, not the sequence's elements. Including the index value requires adding it as a sequence element property and then projecting the desired element properties, including the index value. The Where operator is optional in the following examples:

#### C# 3.0

```
var query5 = CustomerList.Select((cust, index) => new {cust, index})  
    .Where(c => c.cust.Country == "USA" && c.index > 70)  
    .Select(c => new { c.cust.CustomerID, c.cust.CompanyName,  
        c.index });  
  
foreach (var c in query5)  
    sbResult.Append(c.ToString() + "\r\n");
```

#### VB 9.0

```
Dim Query5 = CustomerList.Select(Function(Cust, Index) _  
    New With {Cust, Index}) _  
    .Where(Function(c) _  
        c.Cust.Country = "USA" And c.Index > 70) _  
    .Select(Function(c) New With _  
        {c.Cust.CustomerID, c.Cust.CompanyName, c.Index})  
  
For Each c In Query5  
    sbResult.Append(c.ToString() & vbCrLf  
Next
```

The preceding snippets return the following text (index is Index with the VB 9.0 version):

```
{ CustomerID = SPLIR, CompanyName = Split Rail Beer & Ale, index = 74 }  
{ CustomerID = THEBI, CompanyName = The Big Cheese, index = 76 }  
{ CustomerID = THECR, CompanyName = The Cracker Box, index = 77 }  
{ CustomerID = TRAIH, CompanyName = Trail's Head Gourmet Provisioners, index = 81 }  
{ CustomerID = WHITC, CompanyName = White Clover Markets, index = 88 }
```

### Index Value Expressions with the Let Keyword

You can simplify method call syntax and enable query expressions for queries that need to return index values by combining the IndexOf(member) method with intermediate values stored in a variable assigned with the let/Let keyword, as shown here:

#### C# 3.0

```
var query5 = from cust in CustomerList  
    let index = CustomerList.IndexOf(cust)  
    where cust.Country == "USA" && index > 70  
    select new { cust.CustomerID, cust.CompanyName, index };
```

## Part II: Introducing Language Integrated Query

---

### VB 9.0

```
Dim Query5 = From c In CustomerList _
    Let Index = CustomerList.IndexOf(c) _
    Where c.Country = "USA" And Index > 70 _
    Select New With {c.CustomerID, c.CompanyName, Index}
```

These snippets return the same five rows as the two examples of the preceding section.

## SelectMany

The `SelectMany` operator implements cross and outer joins, and flattens sequences into elements that resemble rows of relational views based on joins. `SelectMany` returns from the source object a many-to-one sequence having the set of properties specified by the `selector` that's passed to the lambda function. Enumerating the returned sequence maps each element to a sequence, which it also enumerates.

### Basic SelectMany Implementation for Associated Objects

The basic `SelectMany` operator projects each element of a sequence to an `Enumerable` object and flattens (combines) the resulting sequences into one sequence. These two implementations join a collection's members with its associated object's members:

### C# 3.0

```
public static IEnumerable<TResult> SelectMany<TSource, TResult> (
    IEnumerable<TSource> source,
    Func<TSource, IEnumerable<TResult>> selector
)
```

### VB 9.0

```
Public Shared Function SelectMany(Of TSource, TResult) ( _
    source As IEnumerable(Of TSource), _
    selector As Func(Of TSource, IEnumerable(Of TResult)) _
) As IEnumerable(Of TResult)
```

This implementation emulates either of these nested iterators within the query body:

### C# 3.0

```
foreach (var c in query)
    foreach (var o in c.Orders)
        sbResult.Append(String.Format("{... Expression ...}));
```

### VB 9.0

```
For Each c In Query
    For Each o In c.Orders
        sbResult.Append(String.Format("{... Expression ...}"))
    Next o
Next c
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

The following examples invoke the basic `SelectMany` implementation to expand the `Orders` collection associated with most `Customer` elements:

### C# 3.0

```
var query6 = CustomerList.Where(c => c.Country == "USA")
    .SelectMany(o => o.Orders);

query6 from c in CustomerList
        where c.Country == "USA"
        from o in c.Orders
        select o;

foreach (var o in query6)
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}, "
        "OrderID = {2}\r\n", o.CustomerID, o.Customer.CompanyName, o.OrderID));
```

### VB 9.0

```
Dim Query6 = CustomerList.Where(Function(c) c.Country = "USA") _
    .SelectMany(Function(c) c.Orders)

Query6 = From c In CustomerList _
    Where c.Country = "USA" _
    From o In c.Orders _
    Select o

For Each o In Query6
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}, "
        "OrderID = {2}" & vbCrLf, o.CustomerID, o.Customer.CompanyName, o.OrderID))
Next
```

*Replacing `Select` with `SelectMany` for source sequences with an associated collection returns an `List<T>` instead of `IEnumerable<T>`. The C# and VB compilers translate `Select` query expressions with nested `from/From` clauses to the `SelectMany` version. The preceding examples won't compile if the nested `from o in c.Orders` or `From o In c.Orders` clause is missing and the `Select` alias is `c` rather than `o`.*

The two preceding snippets return the following text (abbreviated):

```
CustomerID = LONEP, CompanyName = Lonesome Pine Restaurant, OrderID = 10307
CustomerID = LONEP, CompanyName = Lonesome Pine Restaurant, OrderID = 10317
CustomerID = OLDWO, CompanyName = Old World Delicatessen, OrderID = 10305
CustomerID = OLDWO, CompanyName = Old World Delicatessen, OrderID = 10338
...
CustomerID = THEBI, CompanyName = The Big Cheese, OrderID = 10310
CustomerID = WHITC, CompanyName = White Clover Markets, OrderID = 10269
CustomerID = WHITC, CompanyName = White Clover Markets, OrderID = 10344
```

Compare this flattened result with the hierarchical output shown in the earlier “Compound Where Expressions” section.

## Part II: Introducing Language Integrated Query

---

### SelectMany Overload for Equi-Joins of Associated Objects

An overload of the basic SelectMany operator projects each element of a sequence to an Enumerable object, flattens the resulting sequences into one sequence, and applies a result selector function to each element of the sequence.

#### C# 3.0

```
public static IEnumerable<TResult> SelectMany<TSource, TCollection, TResult> (
    IEnumerable<TSource> source,
    Func<TSource, IEnumerable<TCollection>> collectionSelector,
    Func<TSource, TCollection, TResult> resultSelector
)
```

#### VB 9.0

```
Public Shared Function SelectMany(Of TSource, TCollection, TResult) ( _
    source As IEnumerable(Of TSource), _
    collectionSelector As Func(Of TSource, IEnumerable(Of TCollection)), _
    resultSelector As Func(Of TSource, TCollection, TResult) _
) As IEnumerable(Of TResult)
```

The compiler translates query expressions with multiple `from` clauses, such as the following, into one or more of the preceding `SelectMany` overloads:

#### C# 3.0

```
var query7 = from c in CustomerList
             from o in OrderList
             where c.Country == "USA" && o.CustomerID == c.CustomerID
             select new { o, c };

// Translates to:
query7 = from c in CustomerList.SelectMany(o => OrderList, (c, o) => new { c, o })
         where (c.c.Country == "USA") && (c.o.CustomerID == c.c.CustomerID)
         select new { c.o, c.c };

foreach (var x in query7)
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}, " +
        "OrderID = {2}\r\n", x.o.CustomerID, x.c.CompanyName, x.o.OrderID));
```

The simplified translation of the `SelectMany` expression isn't even *close* to intuitive, which is one of the reasons that you see very few examples of `SelectMany` invocations for nontrivial source data.

*Chapter 4's "Disassembling and Translating Query Expressions" section explains how to use Red Gate's (formerly Lutz Roeder's) .NET Reflector application to regenerate C# or VB source code from query expression translations compiled to IL in library or executable files.*

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

### VB 9.0

```
Dim Query7 = From c In CustomerList _
    From o In OrderList _
    Where c.Country = "USA" And o.CustomerID = c.CustomerID _
    Select New With {o, c}

Query7 = From c In CustomerList.SelectMany(Function(o) OrderList, _
    Function(c, o) New With {c, o}) -
    Where (c.c.Country = "USA") And (c.o.CustomerID = c.c.CustomerID) _
    Select New With {c.o, c.c}

For Each x In Query7
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}, _
        OrderID = {2}" & vbCrLf, x.c.CustomerID, x.c.CompanyName, x.o.OrderID))
Next
```

The translation of C# to VB# lambda function syntax only serves to further obfuscate the `SelectMany` expression. This is one of the better examples of how query expressions simplify writing LINQ queries. The output from these two examples is the same as that of the preceding section.

*The preceding example uses SQL-89 equi-join syntax and is less efficient than using LINQ SQL-92-style joins for independent collections. The later “Join Operators” section includes examples of method call syntax for joins between independent collections.*

## Partitioning Operators

As mentioned in the earlier “SQOs as Keywords in C# 3.0 and VB 9.0” section, the partitioning operators enable paging operations and `Take(n)` emulates T-SQL’s `TOP(n)` operator.

Here’s generic code to return a specific page of a given size:

### C# 3.0

```
int pageSize
int pageNum
(from s in dataSource select s).Skip((pageNum -1) * pageSize)
    .Take(pageSize)
```

### VB 9.0

```
Dim PageSize As Integer
Dim PageNum As Integer
From s In DataSource
Select s
Skip (PageNum -1) * PageSize
Take PageSize
```

The `TakeWhile` and `SkipWhile` operators have the index overload, which enables the use of the index value in your paging or positioning logic; the `Take` and `Skip` operators aren’t overloaded.

## Part II: Introducing Language Integrated Query

---

### Take

Take returns from the source sequence the number of contiguous elements specified by count.

#### C# 3.0

```
public static IEnumerable<TSource> Take<TSource> (
    IEnumerable<TSource> source,
    int count
)
```

#### VB 9.0

```
Public Shared Function Take(Of TSource) ( _
    source As IEnumerable(Of TSource), _
    count As Integer _
) As IEnumerable(Of TSource)
```

### Skip

In LINQ for Objects, Skip bypasses in the source enumeration the number of elements specified by count and returns the remaining source elements, if any.

#### C# 3.0

```
public static IEnumerable<TSource> Skip<TSource> (
    IEnumerable<TSource> source,
    int count
)
```

#### VB 9.0

```
Public Shared Function Skip(Of TSource) ( _
    source As IEnumerable(Of TSource), _
    count As Integer _
) As IEnumerable(Of TSource)
```

To minimize the number of rows returned from the database server, LINQ to SQL's Skip implementation behaves differently. Chapter 5's "Paging in LINQ to SQL" and "Paging with the LINQDataSource Control" sections describe the T-SQL statements sent by Take and Skip to SQL Server 200x.

### Skip/Take Example

The following snippets return the first two of the last three U.S. Customer elements from the CustomerList collection:

#### C# 3.0

```
var query8 = CustomerList
    .Select((cust, index) => new { cust, index })
    .Where(c => c.cust.Country == "USA")
    .Select(c => new { c.cust.CustomerID, c.cust.CompanyName, c.index })
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

```
.Skip(10)
    .Take(2);

query8 = (from c in CustomerList
          where c.Country == "USA"
          let index = CustomerList.IndexOf(c)
          select new {c.CustomerID, c.CompanyName, index})
        .Skip(10)
        .Take(2);

foreach (var c in query8)
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}, " +
        "index = {2}\r\n", c.CustomerID, c.CompanyName, c.index));
```

### VB 9.0

```
Dim query8 = CustomerList.Select(Function(Cust, Index) _
    New With {Cust, Index}) _
    .Where(Function(c) c.Cust.Country = "USA") _
    .Select(Function(c) New With _
        {c.Cust.CustomerID, c.Cust.CompanyName, c.Index}) _
    .Skip(10) _
    .Take(2)

query8 = From c In CustomerList _
    Where c.Country = "USA" _
    Let Index = CustomerList.IndexOf(c) _
    Select New With {c.CustomerID, c.CompanyName, Index} _
    Skip 10 _
    Take 2

For Each c In Query8
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}, " & _
        "Index = {2}" + vbCrLf, c.CustomerID, c.CompanyName, c.Index))
Next
```

The preceding examples are based on Query5 from the earlier “Multiple Select Projection Expression with Index Value” section. Here’s the output:

```
CustomerID = THECR, CompanyName = The Cracker Box, Index = 77
CustomerID = TRAIH, CompanyName = Trail's Head Gourmet Provisioners, Index = 81
```

## TakeWhile

`TakeWhile` returns from the source sequence a set of contiguous elements while the predicate expression remains true.

### C# 3.0

```
public static IEnumerable<TSource> TakeWhile<TSource> (
    IEnumerable<TSource> source,
    Func<TSource, bool> predicate
)
```

## Part II: Introducing Language Integrated Query

---

### VB 9.0

```
Public Shared Function TakeWhile(Of TSource) ( _
    source As IEnumerable(Of TSource), _
    predicate As Func(Of TSource, Boolean) _
) As IEnumerable(Of TSource)
```

## SkipWhile

SkipWhile bypasses elements of source sequence, while the predicate expression remains true and returns the remaining source elements, if any.

### C# 3.0

```
public static IEnumerable<TSource> SkipWhile<TSource> ( 
    IEnumerable<TSource> source,
    Func<TSource, bool> predicate
)
```

### VB 9.0

```
Public Shared Function SkipWhile(Of TSource) ( _
    source As IEnumerable(Of TSource), _
    predicate As Func(Of TSource, Boolean) _
) As IEnumerable(Of TSource)
```

## SkipWhile/TakeWhile Example

Following are examples that use method call syntax to return the first element of a U.S. Customer and contiguous elements thereafter, if any.

### C# 3.0

```
var query9 = CustomerList.Select((cust, index) => new { cust, index })
    // .OrderBy(c => c.cust.Country)
    .SkipWhile(c => c.cust.Country != "USA")
    .TakeWhile(c => c.cust.Country == "USA")
    .Select(c => new { c.cust.CustomerID, c.cust.CompanyName, c.index });

foreach (var c in query9)
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}, " +
        "index = {2}\r\n", c.CustomerID, c.CompanyName, c.index));
```

Uncommenting the .OrderBy(c => c.cust.Country) expression returns elements for all U.S. customers.

### VB 9.0

```
Dim Query9 = CustomerList.Select(Function(Cust, Index) _
    New With {Cust, Index}) _
    .SkipWhile(Function(c) c.Cust.Country <> "USA") _
    .TakeWhile(Function(c) c.Cust.Country = "USA") _
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

```
.Select(Function(c) New With {c.Cust.CustomerID, _  
    c.Cust.CompanyName, c.Index})  
  
For Each c In Query9  
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}, " & _  
        "Index = {2}" + vbCrLf, c.CustomerID, c.CompanyName, c.Index))
```

Both preceding examples return the following element:

```
CustomerID = GREAL, CompanyName = Great Lakes Food Market, Index = 31
```

## Join Operators

The two Join operators (`Join` and `GroupJoin`) create equi-joins of two independent objects based on equal values of key properties. The operators use VB 9.0 query expression syntax similar to that of the SQL-92 `JOIN` clause, as in the following:

### VB 9.0

```
From Alias1 In Collection1  
[Group] Join Alias2 In Collection2 On Alias1.Field1 Equals Alias2.Field2  
    [Into Sequence2]  
[[Group] Join Alias3 In Collection3 On Alias2.Field2 Equals Alias3.Field3  
    [Into Sequence3]]  
]  
...  
[Where Expression]  
Select Expression
```

The `Equals` operator replaces the equal sign (=) of the `On` clause to signify that object equality is tested in the nested loop that compares values of outer and inner keys. LINQ joins don't support other SQL-92 compliant comparison operators, such as `>`, `<`, or `<>`. The `Join` and `GroupJoin` operators have overloads that enable customizing the comparer with a special implementation of the `IEqualityComparer(Of TKey)` interface. This book's `Join` examples assume the default implementation of `IEqualityComparer(Of TKey)`, so the sections that follow don't include method signatures for the overloaded method.

A C# `join ... into` clause causes the language compilers to translate `join` to `GroupJoin`, which is similar in effect to translation of `select/Select` to `SelectMany` for query expressions with nested `from/from` clauses. VB requires the explicit `Group Join` keyphrase with an `Into Group` or `Into identifier = Group` keyphrase to deliver a hierarchical set of sequences with the `GroupJoin` operator. The `GroupJoin` operator outputs hierarchical sequences from independent collections, just as `SelectMany` produces hierarchical sequences from dependent collections (subobjects) in fields.

*According to Microsoft, VB 9.0's `Into Group` or `Into identifier = Group` syntax enables parallel processing of grouping operations, whereas C# 3.0's simple `Into` identifier clause doesn't. Parallel query processing with multicore processors has become increasingly important as additional processor cores compensate for speed increases. Microsoft's declarative language for parallel processing of LINQ queries is called PLINQ. PLINQ is an abbreviation for Parallel LINQ, which was in development when this book was written. Chapter 8 provides an example of PLINQ running on a two-core machine.*

## Part II: Introducing Language Integrated Query

---

Nested loop joins, which are also called *nested iterations*, without indexes (*naïve nested loop joins*) exhibit poor performance with LINQ as well as databases, especially with a large sets of outer and inner rows. Thus you should apply the `Where` operator, when applicable, to both the inner and outer collections to minimize the number of rows iterated. Chapter 4's sections about `Joins` discuss applying `Where` clauses before initiating `Joins` to minimize the sets' size. It's not a common practice to index objects but using indexes can increase performance by orders of magnitude on unavoidably large collections. However, Aaron Erickson's Indexes for Objects (i4o) is a helper class for creating indexed LINQ object, which improves performance greatly with large collections. You can learn more about i4o and download a beta version from <http://www.codeplex.com/i4o>.

## Join

Following are method signatures for the C# 3.0 and VB 9.0 versions of the `join/Join` operator:

### C# 3.0

```
public static IEnumerable<TResult> Join<TOuter, TInner, TKey, TResult> (
    IEnumerable<TOuter> outer,
    IEnumerable<TInner> inner,
    Func<TOuter, TKey> outerKeySelector,
    Func<TInner, TKey> innerKeySelector,
    Func<TOuter, TInner, TResult> resultSelector
)
```

### VB 9.0

```
Public Shared Function Join(Of TOuter, TInner, TKey, TResult) ( _
    outer As IEnumerable(Of TOuter), _
    inner As IEnumerable(Of TInner), _
    outerKeySelector As Func(Of TOuter, TKey), _
    innerKeySelector As Func(Of TInner, TKey), _
    resultSelector As Func(Of TOuter, TInner, TResult) _
) As IEnumerable(Of TResult)
```

Method signatures for `join/Join` operators are more important than for many other SQOs, because it's not immediately evident how to translate query expression syntax for joins into method call syntax. The methods perform an integral element key extraction to generate the `TKey` argument value.

Following are method call and query expression examples of code that perform a left inner join over the `CustomerList` (outer) and `OrderList` (inner):

### C# 3.0

```
var query10 = CustomerList
    .Join(OrderList, c => c.CustomerID, o => o.CustomerID, (c, o) => new
        { c.CustomerID, c.CompanyName, o.OrderID, o.OrderDate })
    .Take(10);

query10 = (from c in CustomerList
            join o in OrderList on c.CustomerID equals o.CustomerID
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

```
        select new { c.CustomerID, c.CompanyName, o.OrderID, o.OrderDate })
        .Take(10);

    foreach (var co in query10)
        sbResult.Append(String.Format("CustID = {0}, Name = {1}, Order = {2}, " +
            "Date = {3:d}\r\n", co.CustomerID, co.CompanyName, co.OrderID,
            co.OrderDate));
```

c.CustomerID is the outerKeySelector, o.CustomerID is the innerKeySelector, and the object initializer is the resultSelector.

### VB 9.0

```
Dim Query10 = CustomerList _
    .Join(OrderList, Function(c) c.CustomerID, Function(o) _
        o.CustomerID, Function(c, o) New With _
        {c.CustomerID, c.CompanyName, o.OrderID, o.OrderDate}) _
    .Take(10)

Query10 = (From c In CustomerList _
    Join o In OrderList On c.CustomerID Equals o.CustomerID _
    Select New With {c.CustomerID, c.CompanyName, o.OrderID, _
        o.OrderDate}) _
    .Take(10)

For Each co In Query10
    sbResult.Append(String.Format("CustID = {0}, Name = {1}, Order = {2}, " & _
        "Date = {3:d}" + vbCrLf, co.CustomerID, co.CompanyName, co.OrderID, _
        co.OrderDate))
Next
```

## GroupJoin

The GroupJoin operator is similar to the SelectMany SQO in that it can generate a hierarchical output that interleaves outer members with a nested IEnumerable<T> collection of inner members. You specify the a GroupJoin operation in a query expression by adding the into/ Into keyword followed by an object identifier to the Join statement. A C# 3.0 query expression that includes the into identifier and contains two or more from clauses causes the language compiler to translate the clauses into a single GroupJoin expression. This is another respect in which the GroupJoin operator is similar to SelectMany.

*As mentioned earlier in the chapter, VB 9.0 includes the Group Join keyphrase, which is required to support the Into Group or Into identifier = Group keyphrase.*

The GroupJoin operator also enables generating left outer joins by adding the DefaultIfEmpty ([DefaultInnerObject]) element operator to add an outer collection element when there is no matching inner collection element. The DefaultInnerObject definition is optional, but its use is recommended.

## Part II: Introducing Language Integrated Query

---

Here are method signatures for the C# 3.0 and VB 9.0 versions of the `GroupJoin` operator:

### C# 3.0

```
public static IEnumerable<TResult> GroupJoin<TOuter, TInner, TKey, TResult> (
    IEnumerable<TOuter> outer,
    IEnumerable<TInner> inner,
    Func<TOuter, TKey> outerKeySelector,
    Func<TInner, TKey> innerKeySelector,
    Func<TOuter, IEnumerable<TInner>, TResult> resultSelector
)
```

### VB 9.0

```
Public Shared Function GroupJoin(Of TOuter, TInner, TKey, TResult) ( _
    outer As IEnumerable(Of TOuter), _
    inner As IEnumerable(Of TInner), _
    outerKeySelector As Func(Of TOuter, TKey), _
    innerKeySelector As Func(Of TInner, TKey), _
    resultSelector As Func(Of TOuter, IEnumerable(Of TInner), TResult) _
) As IEnumerable(Of TResult)
```

The following examples implement a left outer join of `Customer` with `Order` objects with default values supplied to display `Customer` objects that don't have corresponding `Order` objects:

### C# 3.0

```
// Populate the default order instance
Order emptyOrder = new Order { OrderID = 0, CustomerID = "XXXXX", EmployeeID = 0,
OrderDate = DateTime.Parse("1/1/1900"), RequiredDate = DateTime.Parse("1/1/1900"),
ShipVia = 0, Freight = 0M, ShipName = "Default Order" };

var query11 = from c in CustomerList
              where c.Country == "USA"
              join o in OrderList on c.CustomerID equals o.CustomerID into co
              from d in co.DefaultIfEmpty(emptyOrder)
              select new { c, d };

foreach (var c in query11)
    sbResult.Append(String.Format("CustID = {0}, Name = {1}, OrderID = {2},
        OrderDate = {3:d}\r\n", c.c.CustomerID, c.c.CompanyName, c.d.OrderID,
        c.d.OrderDate));
```

Providing a default `emptyOrder` instance is a good LINQ programming practice. You'll probably encounter runtime errors if your inner object has non-nullable value types and you don't provide a default instance populated with default values as the `DefaultIfEmpty` operator's argument.

*Many LINQ outer join examples from whitepapers and blogs haven't been tested with real-world sequences that generate missing inner elements with value-type fields or have missing iterator code. You haven't tested left out join code until you iterate the query result.*

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

### VB 9.0

```
Dim EmptyOrder = New Order With {.OrderID = 0, .CustomerID = "XXXXX", _
    .EmployeeID = 0, .OrderDate = DateTime.Parse("1/1/1900"), _
    .RequiredDate = #1/1/1900#, .ShipVia = 0, .Freight = 0, _
    .ShipName = "Default Order"}

Dim Query11 = From c In CustomerList _
    Where c.Country = "USA" _
    Group Join o In OrderList On c.CustomerID Equals _
        o.CustomerID Into Group _
    From d In Group.DefaultIfEmpty(EmptyOrder) _
    Select New With {c, d}

For Each c In Query11
    sbResult.Append(String.Format("CustID = {0}, Name = {1}, OrderID = {2}, " & _
        "OrderDate = {3:d}" & vbCrLf, c.c.CustomerID, c.c.CompanyName, & _
        c.d.OrderID, c.d.OrderDate))
Next
```

*According to its IntelliSense tooltip, you “Use ‘Group’ to specify that a group named ‘Group’ should be created.” Group is the only identifier permitted as the <group alias> assignment for VB 9.0’s Into keyword. Alternatively, you can substitute one of aggregate operators for Group.*

Both examples deliver the same 22-line result (rows abbreviated and CompanyName truncated to 20 characters due to publishing limitations):

```
CustID = GREAL, Name = Great Lakes Food Ma, OrderID = 0, OrderDate = 1/1/1900
CustID = HUNGC, Name = Hungry Coyote Impor, OrderID = 0, OrderDate = 1/1/1900
CustID = LAZYK, Name = Lazy K Kountry Stor, OrderID = 0, OrderDate = 1/1/1900
CustID = LETSS, Name = Let's Stop N Shop, OrderID = 0, OrderDate = 1/1/1900
CustID = LONEP, Name = Lonesome Pine Resta, OrderID = 10307, OrderDate = 9/17/2006
CustID = LONEP, Name = Lonesome Pine Resta, OrderID = 10317, OrderDate = 9/30/2006
CustID = OLDWO, Name = Old World Delicates, OrderID = 10305, OrderDate = 9/13/2006
CustID = OLDWO, Name = Old World Delicates, OrderID = 10338, OrderDate = 10/25/2006
CustID = RATTC, Name = Rattlesnake Canyon , OrderID = 10262, OrderDate = 7/22/2006
...
CustID = SAVEA, Name = Save-a-lot Markets, OrderID = 10324, OrderDate = 10/8/2006
CustID = SPLIR, Name = Split Rail Beer & A, OrderID = 10271, OrderDate = 8/1/2006
CustID = SPLIR, Name = Split Rail Beer & A, OrderID = 10329, OrderDate = 10/15/2006
CustID = THEBI, Name = The Big Cheese, OrderID = 10310, OrderDate = 9/20/2006
CustID = THECR, Name = The Cracker Box, OrderID = 0, OrderDate = 1/1/1900
CustID = TRAIH, Name = Trail's Head Gourme, OrderID = 0, OrderDate = 1/1/1900
CustID = WHITC, Name = White Clover Market, OrderID = 10269, OrderDate = 7/31/2006
CustID = WHITC, Name = White Clover Market, OrderID = 10344, OrderDate = 11/1/2006
```

Great Lakes Food Markets, Lazy K Kountry Store, Let's Stop and Shop, Inc., The Big Cheese, The Cracker Box, and Trail's Head Gourmet have no orders in the first 100 Order instances.

*Chapter 4 has several sections devoted to advanced uses of the GroupJoin SQO and related keywords.*

# Concatenation Operator: Concat

The Concat operator concatenates two sequences into one. Following are its method signatures:

### C# 3.0

```
public static IEnumerable<TSource> Concat<TSource> (
    IEnumerable<TSource> first,
    IEnumerable<TSource> second
)
```

### VB 9.0

```
Public Shared Function Concat(Of TSource) ( _
    first As IEnumerable(Of TSource), _
    second As IEnumerable(Of TSource) _
) As IEnumerable(Of TSource)
```

The following examples combine sequences for U.S., Canadian, and Mexican customers into one sequence of North American customers, which preserves the addition sequence:

### C# 3.0

```
var Query12 = CustomerList.Where(c => c.Country == "USA")
    .Concat(CustomerList.Where(c => c.Country == "Canada"))
    .Concat(CustomerList.Where(c => c.Country == "Mexico"));

Query12 = (from c in CustomerList
    where c.Country == "USA"
    select c)
    .Concat((from c in CustomerList
        where c.Country == "Canada"
        select c))
    .Concat((from c in CustomerList
        where c.Country == "Mexico"
        select c));

foreach (var c in Query12)
    sbResult.Append(String.Format("CustID = {0}, Name = {1}, Country = {2}\r\n",
        c.CustomerID, c.CompanyName, c.Country));
```

### VB 9.0

```
Dim Query12 = CustomerList.Where(Function(c) c.Country = "USA") _
    .Concat(CustomerList.Where(Function(c) c.Country = "Canada")) _
    .Concat(CustomerList.Where(Function(c) c.Country = "Mexico"))

Query12 = (From c In CustomerList _
    Where c.Country = "USA") _
    .Concat((From c In CustomerList _
        Where c.Country = "Canada") _
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

```
.Concat((From c In CustomerList _  
Where c.Country = "Mexico")))  
  
For Each c In Query12  
    sbResult.Append(String.Format("CustID = {0}, Name = {1}, Country = {2}" & _  
        vbCrLf, c.CustomerID, c.CompanyName, c.Country))  
Next
```

Following is the output (with U.S. customers rows abbreviated):

```
CustID = GREAL, Name = Great Lakes Food Market, Country = USA  
...  
CustID = WHITC, Name = White Clover Markets, Country = USA  
CustID = BOTTM, Name = Bottom-Dollar Markets, Country = Canada  
CustID = LAUGB, Name = Laughing Bacchus Wine Cellars, Country = Canada  
CustID = MEREP, Name = Mère Paillarde, Country = Canada  
CustID = ANATR, Name = Ana Trujillo Emparedados y helados, Country = Mexico  
CustID = ANTON, Name = Antonio Moreno Taquería, Country = Mexico
```

It would be quite difficult to generate this sequence that lists customers alphabetically within arbitrary groups of countries by SQL-like set operators.

*Avoid the use of the Concat () operator with collections having many members. Because of language restrictions Concat () performs m \* n operations instead of m + n operations, where m and n are the number of members of the two collections. For more information on this issue, see Wes Dyer's "All About Iterators" blog post (<http://blogs.msdn.com/wesdyer/archive/2007/03/23/all-about-iterators.aspx>).*

## Ordering Operators

The ordering operators, with the exception of `Reverse`, behave identically to their SQL counterparts. The `ThenBy` and `ThenByDescending` operators differ from the `OrderBy` and `OrderByDescending` versions by operating on the `IOrderedEnumerable<T>` type. All ordering operators except `Reverse` have overloads that enable substitution of a custom `IComparer< TKey >`. The methods perform the required element key extraction operation to obtain `TKey` argument values.

*The OrderBy and ThenBy operators provide an unstable sort for the LINQ pattern. An unstable sort is an ordering that doesn't guarantee preservation of the original order of multiple elements having the same sort key value. However, but LINQ to Objects provides a stable sort and sorts on LINQ to SQL or LINQ to Entities are implemented by the generated T-SQL or SQL statement.*

Following are the method signatures for the `OrderBy` SQO:

### C# 3.0

```
public static IOrderedEnumerable<TSource> OrderBy<TSource, TKey> (  
    IEnumerable<TSource> source,  
    Func<TSource, TKey> keySelector  
)
```

## Part II: Introducing Language Integrated Query

---

### VB 9.0

```
Public Shared Function OrderBy(Of TSource, TKey) ( _  
    source As IEnumerable(Of TSource), _  
    keySelector As Func(Of TSource, TKey) _  
) As IOrderedEnumerable(Of TSource)
```

The keywords for the `OrderBy` SQO are `orderby` in C# and `Order By` in VB.

## **OrderByDescending**

For completeness, here are the method signatures for the `OrderByDescending` SQO:

### C# 3.0

```
public static IOrderedEnumerable<TSource> OrderByDescending<TSource, TKey> ( _  
    IEnumerable<TSource> source,  
    Func<TSource, TKey> keySelector  
)
```

### VB 9.0

```
Public Shared Function OrderByDescending(Of TSource, TKey) ( _  
    source As IEnumerable(Of TSource), _  
    keySelector As Func(Of TSource, TKey) _  
) As IOrderedEnumerable(Of TSource)
```

Differences from `OrderBy` are emphasized.

## **ThenBy**

Here's the same exercise for `ThenBy`:

### C# 3.0

```
public static IOrderedEnumerable<TSource> ThenBy<TSource, TKey> ( _  
    IOrderedEnumerable<TSource> source,  
    Func<TSource, TKey> keySelector  
)
```

### VB 9.0

```
Public Shared Function ThenBy(Of TSource, TKey) ( _  
    source As IOrderedEnumerable(Of TSource), _  
    keySelector As Func(Of TSource, TKey) _  
) As IOrderedEnumerable(Of TSource)
```

The language compiler translates second or more uses of `orderby/Order By` in a query expression to `ThenBy`.

### **ThenByDescending**

And ThenByDescending:

#### **C# 3.0**

```
public static IObservable<TSource> ThenByDescending<TSource, TKey> (
    IObservable<TSource> source,
    Func<TSource, TKey> keySelector
)
```

#### **VB 9.0**

```
Public Shared Function ThenByDescending(Of TSource, TKey) ( _
    source As IObservable(Of TSource), _
    keySelector As Func(Of TSource, TKey) _
) As IObservable(Of TSource)
```

The language compiler translates second or more uses of `orderby ... descending`/`Order By ... Descending` in a query expression to `ThenByDescending`.

### **Reverse**

Reverse — as you would expect — reverses the order of the elements and returns an `IEnumerable<T>`, not an `IObservable<T>`, and doesn't take `TKey` as a lambda function argument.

#### **C# 3.0**

```
public static IEnumerable<TSource> Reverse<TSource> (
    IEnumerable<TSource> source
)
```

#### **VB 9.0**

```
Public Shared Function Reverse(Of TSource) ( _
    source As IEnumerable(Of TSource) _
) As IEnumerable(Of TSource)
```

### **Ordering Operator Examples**

It's difficult to improve on the C# examples from the "Standard Query Operators" whitepaper without resorting to trivial modifications, so here are three examples that are modified only to make the code compatible with this chapter's sample objects:

#### **C# 3.0**

```
var query13 = ProductList.OrderBy(p => p.CategoryID)
    .ThenByDescending(p => p.UnitPrice)
    .ThenBy(p => p.ProductName);

query13 = from p in ProductList
```

## Part II: Introducing Language Integrated Query

---

```
orderby p.CategoryID, p.UnitPrice descending, p.ProductName
select p;

foreach (var p in query13)
    sbResult.Append(String.Format("CategoryID = {0}, UnitPrice = {1:c}, " +
        Name = {2}\r\n", p.CategoryID, p.UnitPrice, p.ProductName));

var query14 = ProductList.Where(p => p.Category.CategoryName == "Beverages")
    .OrderBy(p => p.ProductName,
        StringComparer.CurrentCultureIgnoreCase);

foreach (var p in query14)
    sbResult.Append(String.Format("ProductID = {0}, UnitPrice = {1:c}, " &
        "Name = {2}\r\n", p.ProductID, p.UnitPrice, p.ProductName));
```

Ordering by `ProductName` in `query13` isn't meaningful because no two products in a category have the same unit price. The custom `StringComparer` implements `IComparer`, `IEqualityComparer`, `Comparer<string>`, and `IEqualityComparer<string>`, and lets you choose `InvariantCulture`, `InvariantCultureIgnoreCase`, `CurrentCulture`, `CurrentCultureIgnoreCase`, `Ordinal`, or `OrdinalIgnoreCase` as the sort collation.

### VB 9.0

```
Dim Query13 = ProductList.OrderBy(Function(p) p.CategoryID) _
    .ThenByDescending(Function(p) p.UnitPrice) _
    .ThenBy(Function(p) p.ProductName)

Query13 = From p In ProductList _
    Order By p.CategoryID, p.UnitPrice Descending, p.ProductName

For Each p In Query13
    sbResult.Append(String.Format("CategoryID = {0}, UnitPrice = {1:c}, " & _
        "Name = {2}" & vbCrLf, p.CategoryID, p.UnitPrice, p.ProductName))
Next p

Dim Query14 = ProductList.Where(Function(p) p.Category.CategoryName = _
    "Beverages") _
    .OrderBy(Function(p) p.ProductName, _
        StringComparer.CurrentCultureIgnoreCase)

For Each p In Query14
    sbResult.Append(String.Format("ProductID = {0}, UnitPrice = {1:c}, " & _
        "Name = {2}" & vbCrLf, p.ProductID, p.UnitPrice, p.ProductName))
Next p
```

If you add `Select p` to the VB query expression version of `Query13`, you'll receive a runtime exception because of a type conflict between `IOrderedEnumerable(Of Product)` from the method call syntax version and the anonymous `<SelectIterator>` type returned by `Select p`. The exception doesn't occur with the C# version; `select p` returns `IOrderedEnumerable<Product>`.

*The sample code shows the output of the preceding queries in a text box.*

# Grouping Operator: GroupBy

The `GroupBy` SQO performs a function similar to the SQL's `GROUP BY` clause and shares some project features with the `SelectMany` and `GroupJoin` operators. The `group identifier by` query expression clause returns a sequence of `IGrouping` objects. The sequence contains items that match the group's key value. The `GroupBy` method performs the required element key extraction operation to obtain `TKey` argument values. The `IGrouping<TKey, TSource>` type is an `IEnumerable<TSource>` with an added key value for each element.

The four `GroupBy` operator overloads perform the following operations:

1. Group the elements of a sequence in accordance with `keySelector` values
2. Group the elements of a sequence in accordance with `keySelector` values and selects the resulting elements with an `elementSelector` function
3. Group the elements of a sequence in accordance with `keySelector` values and generates a `resultSelector` value from each group and its key
4. Group the elements of a sequence in accordance with `keySelector`, generates a `resultSelector` value from each group and its key and selects the resulting elements with an `elementSelector` function

### C# 3.0

```
public static IEnumerable<IGrouping<TKey, TSource>> GroupBy<TSource, TKey> (
    IEnumerable<TSource> source,
    Func<TSource, TKey> keySelector
)

public static IEnumerable<IGrouping<TKey, TElement>> GroupBy<TSource, TKey,
TElement> (
    IEnumerable<TSource> source,
    Func<TSource, TKey> keySelector,
    Func<TSource, TElement> elementSelector
)

public static IEnumerable<TResult> GroupBy<TSource, TKey, TResult> (
    IEnumerable<TSource> source,
    Func<TSource, TKey> keySelector,
    Func<TKey, IEnumerable<TSource>, TResult> resultSelector
)

public static IEnumerable<TResult> GroupBy<TSource, TKey, TElement, TResult> (
    IEnumerable<TSource> source,
    Func<TSource, TKey> keySelector,
    Func<TSource, TElement> elementSelector,
    Func<TKey, IEnumerable<TElement>, TResult> resultSelector
)
```

*Overloads are presented in sequence for each language to make it easier to distinguish the difference between the methods' signatures.*

## Part II: Introducing Language Integrated Query

---

### VB 9.0

```
Public Shared Function GroupBy(Of TSource, TKey) ( _
    source As IEnumerable(Of TSource), _
    keySelector As Func(Of TSource, TKey) _
) As IEnumerable(Of IGrouping(Of TKey, TSource))  
  
Public Shared Function GroupBy(Of TSource, TKey, TElement) ( _
    source As IEnumerable(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    elementSelector As Func(Of TSource, TElement) _
) As IEnumerable(Of IGrouping(Of TKey, TElement))  
  
Public Shared Function GroupBy(Of TSource, TKey, TResult) ( _
    source As IEnumerable(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    resultSelector As Func(Of TKey, IEnumerable(Of TSource), TResult) _
) As IEnumerable(Of TResult)  
  
Public Shared Function GroupBy(Of TSource, TKey, TElement, TResult) ( _
    source As IEnumerable(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    elementSelector As Func(Of TSource, TElement), _
    resultSelector As Func(Of TKey, IEnumerable(Of TElement), TResult) _
) As IEnumerable(Of TResult)
```

*There are an additional three overloads for each operator that enable you to specify a custom `IEqualityComparer<TKey>`. All examples in this book use `EqualityComparer<TKey>.Default`.*

## GroupBy with Method Call Syntax

You can perform a useful `GroupBy` operation with a single SQO, but ordering the group key and member elements improves readability, as demonstrated by these two method call syntax examples that group the `CustomerList` members by their `Country` property value:

### C# 3.0

```
var query15 = CustomerList.GroupBy(c => c.Country);  
  
query15 = CustomerList.OrderBy(c => c.Country)
    .ThenBy(c => c.CustomerID)
    .GroupBy(c => c.Country);  
  
foreach (var g in query15) {
    sbResult.Append(g.Key + "\r\n";
    foreach (var c in g)
        sbResult.Append(String.Format("    CustomerID = {0}, CompanyName = {1}, " +
            "City = {2}\r\n", c.CustomerID, c.CompanyName, c.City));
}
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

### VB 9.0

```
Dim Query15 = CustomerList.GroupBy(Function(c) c.Country)

    Query15 = CustomerList.OrderBy(Function(c) c.Country) _
        .ThenBy(Function(c) c.CustomerID) _
        .GroupBy(Function(c) c.Country)

For Each g In Query15
    sbResult.Append(g.Key & vbCrLf)
    For Each c In g
        sbResult.Append(String.Format("    CustomerID = {0}, CompanyName = {1}, " & _
            "City = {2}" & vbCrLf, c.CustomerID, c.CompanyName, c.City))
    Next
Next
```

## GroupBy with Query Expression Syntax

Following are `group...by...into` and `Group By...Into` examples that use query expression syntax, which differs considerably between C# 3.0 and VB 9.0, to group Customer objects by Country:

### C# 3.0

```
var query16 = from c in CustomerList
              orderby c.Country, c.CustomerID
              group c by c.Country into g
              select new { Country = g.Key, Group = g };

foreach (var g in query16) {
    sbResult.Append(g.Group + "\r\n");
    foreach (var c in g.Customer)
        sbResult.Append(String.Format("    CustomerID = {0}, CompanyName = {1}, " +
            "City = {2}/r/n", c.CustomerID, c.CompanyName, c.City));
}
```

Despite the fact that it's legal to terminate query expressions with `group` or `select` statements, you'll receive a compile-time error if you don't finish with `select new { ... };` in C# code.

### VB 9.0

```
Dim Query16A = From c In CustomerList _
    Order By c.Country, c.CustomerID _
    Group c By g = c.Country Into Group _
    'Select Country, Group 'optional

For Each x In Query16A
    sbResult.Append(x.Country & vbCrLf)
    For Each c In x.Group
        sbResult.Append(String.Format("    CustomerID = {0}, CompanyName = {1}, " & _
            "City = {2}" & vbCrLf, c.CustomerID, c.CompanyName, c.City))
    Next c
Next x
```

## Part II: Introducing Language Integrated Query

---

VB 9.0's implementation of the `Group identifier By ... Into` clause is the same as that for the `Group Join` clause, but the `Group By` clause lets you add an identifier for the Key value also, as in:

### VB 9.0

```
Dim Query16C = From c In CustomerList _
    Order By c.Country, c.CustomerID _
    Group c By c = c.Country Into g = Group _
    'Select c, g 'optional

For Each x In Query16C
    sbResult.Append(x.c & vbCrLf)
    For Each c In x.g
        sbResult.Append(String.Format("    CustomerID = {0}, " & _
            "CompanyName = {1}, City = {2}" & vbCrLf, _
            c.CustomerID, c.CompanyName, c.City))
    Next c
Next x
```

*Both VB 9.0 Group By expressions use an implicit Select clause.*

Here are the first and last parts of the result from executing the preceding three queries:

```
Argentina
CustomerID = CACTU, CompanyName = Cactus Comidas, City = Buenos Aires
CustomerID = OCEAN, CompanyName = Océano Atlántico Ltda., City = Buenos Aires
CustomerID = RANCH, CompanyName = Rancho grande, City = Buenos Aires
...
Venezuela
CustomerID = GROSR, CompanyName = GROSELLA-Restaurante, City = Caracas
CustomerID = HILAA, CompanyName = HILARION-Abastos, City = San Cristóbal
CustomerID = LILAS, CompanyName = LILA-Supermercado, City = Barquisimeto
CustomerID = LINOD, CompanyName = LINO-Delicateses, City = I. de Margarita
```

Chapter 4 contains several examples of complex grouping and use of aggregate values with grouped sequences.

## Set Operators

With the exception of `Distinct`, set operators — `Distinct`, `Union`, `Intersect`, and `Except` — combine two sequences into one. Set operators emulate T-SQL's corresponding `DISTINCT` keyword and the `UNION`, `INERSECT`, and `EXCEPT` operators.

### **Distinct**

The `Distinct` SQO removes elements with duplicate values from a sequence. Following are the method signatures:

#### **C# 3.0**

```
public static IEnumerable<TSource> Distinct<TSource> (
    IEnumerable<TSource> source
)
```

#### **VB 9.0**

```
Public Shared Function Distinct(Of TSource) ( _
    source As IEnumerable(Of TSource) _
) As IEnumerable(Of TSource)
```

All set operators have overloads to enable substituting a custom `IEqualityComparer<TSource>` object for the default comparer implementation.

The following two examples return a list of unique country names from the `CustomerList` collection's `Country` field:

#### **C# 3.0**

```
var query17 = (from c in CustomerList
               orderby c.Country
               select c.Country).Distinct();

foreach (var c in query17)
    sbResult.Append(String.Format("Country = {0}\r\n", c));
```

#### **VB 9.0**

```
Dim Query17 = (From c In CustomerList _
                Order By c.Country _
                Select c.Country).Distinct()

For Each c In Query17
    sbResult.Append(String.Format("Country = {0}" & vbCrLf, c))
Next c
```

### **Union**

The `Union` SQO combines two sequences of the same structure into one without regard to matching values. Here are the method signatures:

#### **C# 3.0**

```
public static IEnumerable<TSource> Union<TSource> (
    IEnumerable<TSource> first,
    IEnumerable<TSource> second
)
```

## Part II: Introducing Language Integrated Query

---

### VB 9.0

```
Public Shared Function Union(Of TSource) ( _  
    first As IEnumerable(Of TSource), _  
    second As IEnumerable(Of TSource) _  
) As IEnumerable(Of TSource)
```

Chapter 1's "Creating Explicit Joins with LINQ Join . . . On Expressions" section has a LINQ to SQL example that sends T-SQL join statements to SQL Server 2005 Express to generate source sequences from the Northwind Customers and Suppliers lists. The following two examples perform the same join on `CustomerList` and `SupplierList` collections:

### C# 3.0

```
var query18 = (  
    from c in CustomerList  
    join s in SupplierList on c.City equals s.City  
    select new { c.City, c.CompanyName, Type = "Customer" })  
.Union(  
    from c in CustomerList  
    join s in SupplierList on c.City equals s.City  
    select new { s.City, s.CompanyName, Type = "Supplier" })  
.OrderBy(c => c.City)  
.ThenBy(c => c.CompanyName);  
  
// Append list head  
sbResult.Append("City\tCompany Name\tRelationship\r\n");  
// Append query results  
foreach (var u in query18){  
    string strFormat = "{0}\t{1}\t";  
    if (u.CompanyName.Length < 20) strFormat += "\t";  
    strFormat += "{2}\r\n";  
    sbResult.Append(string.Format(strFormat, u.City, u.CompanyName, u.Type));  
}
```

In this case, however, you must chain `OrderBy` and `ThenBy` operators (emphasized) to sort the union of the two sequences by city and company name because the `Customer` sequence is ordered by `CustomerID` and the `Supplier` sequence is sorted by `SupplierID`.

### VB 9.0

```
Dim Query18 = (  
    From c In CustomerList _  
    Join s In SupplierList On c.City Equals s.City _  
    Select New With {c.City, c.CompanyName, .Type = "Customer"}) _  
.Union( _  
    From c In CustomerList _  
    Join s In SupplierList On c.City Equals s.City _  
    Select New With {s.City, s.CompanyName, .Type = "Supplier"}) _
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

```
.OrderBy(Function(c) c.City)
.ThenBy(c => c.CompanyName)

' Append list header
sbResult.Append("City" & vbTab & "Company Name" & vbTab & vbTab & "Type" & vbCrLf)
' Append query results
For Each u In Query18
    Dim strFormat As String = "{0}" & vbTab & "{1}" & vbTab
    If u.CompanyName.Length() < 20 Then
        strFormat &= vbTab
    End If
    strFormat &= "{2}" & vbCrLf
    sbResult.Append(String.Format(strFormat, u.City, u.CompanyName, u.Type))
Next
```

The output of the preceding queries is (abbreviated):

City	Company Name	Type
Berlin	Alfreds Futterkiste	Customer
Berlin	Heli Süßwaren GmbH & Co. KG	Supplier
...		
Sao Paulo	Queen Cozinha	Customer
Sao Paulo	Refrescos Americanas LTDA	Supplier
Sao Paulo	Tradição Hipermercados	Customer

## Intersect

The `Intersect` SQO returns a sequence of elements that the first and second sequences contain; its method signature is the same as `Union`'s except for the method name.

Following are examples of the use of the `Intersect` SQO with the `ProductsList` collection:

### C# 3.0

```
var query19 = (
    from p in ProductList
    where p.CategoryID == 1 || p.CategoryID == 2
    select p)
.Intersect(
    from p in ProductList
    where p.CategoryID == 2 || p.CategoryID == 3
    select p)
.OrderBy(p => p.ProductName);

foreach (var p in query19)
    sbResult.Append(String.Format("CategoryID = {0}, ProductID = {1}, " +
        "ProductName = {2}\r\n", p.CategoryID, p.ProductID, p.ProductName));
```

## Part II: Introducing Language Integrated Query

---

### VB 9.0

```
Dim Query19 = ( _  
    From p In ProductList _  
    Where p.CategoryID = 1 Or p.CategoryID = 2 _  
        Select p) _  
.Intersect( _  
    From p In ProductList _  
    Where p.CategoryID = 2 Or p.CategoryID = 3 _  
        Select p) _  
.OrderBy(Function(p) p.ProductName)  
  
For Each p In Query19  
    sbResult.Append(String.Format("CategoryID = {0}, ProductID = {1}, " + _  
        "ProductName = {2}" & vbCrLf, p.CategoryID, p.ProductID, p.ProductName))  
Next p
```

Here's the abbreviated output from the preceding two `Intersect` queries.

```
CategoryID = 2, ProductID = 3, ProductName = Aniseed Syrup  
CategoryID = 2, ProductID = 4, ProductName = Chef Anton's Cajun Seasoning  
...  
CategoryID = 2, ProductID = 61, ProductName = Sirop d'éralbe  
CategoryID = 2, ProductID = 63, ProductName = Vegie-spread
```

Products in category 2 (Condiments) are common to the two sequences.

### Except

The `Except` SQO returns a sequence of elements that the first sequence contains but the second sequence doesn't contain; its method signature also is the same as `Union`'s except for the method name.

The preceding two examples with the `Intersect` SQO changed to the `Except` SQO produce this abbreviated output:

```
CategoryID = 1, ProductID = 1, ProductName = Chai  
CategoryID = 1, ProductID = 2, ProductName = Chang  
...  
CategoryID = 1, ProductID = 34, ProductName = Sasquatch Ale  
CategoryID = 1, ProductID = 35, ProductName = Steeleye Stout
```

The query returns products in category 1 (Beverages) only because products from categories 1 and 2 are in the first sequence, categories 2 and 3 are in the second sequence, and the first sequence doesn't contain category 3.

## Conversion operators

Conversion standard query operators change the type of source collections.

### AsEnumerable

The `AsEnumerable` SQO hides a source `IEnumerable<T>` object's local methods for public query operators that would override application of LINQ SQO's. If the source object has no methods, invoking `AsEnumerable` returns the source object. Here are the C# 3.0 and VB 9.0 method signatures:

#### C# 3.0

```
public static IEnumerable<TSource> AsEnumerable<TSource> (
    IEnumerable<TSource> source
)
```

#### VB 9.0

```
Public Shared Function AsEnumerable(Of TSource) ( _
    source As IEnumerable(Of TSource) _
) As IEnumerable(Of TSource)
```

The primary use of `AsEnumerable` is to enable collections that don't implement `IEnumerable<T>` directly, such as ADO.NET `DataTable` objects, to be used as LINQ data sources. Therefore, use of the `AsEnumerable` SQO is uncommon in LINQ to Objects queries. Chapter 1's "Strongly Typed DataSets" and "Untyped DataSets" sections provide introductory C# 3.0 and VB 9.0 code samples that require applying the `AsEnumerable()` method. Chapter 6 provides additional details on the use of the `AsEnumerable` SQO.

The `AsEnumerable` SQO supports deferred query execution by an iterator and does not force immediate execution.

### AsQueryable

The default `AsQueryable` SQO implementation wraps a generic `IEnumerable<T>` type with a generic `IQueryable<T>` type:

#### C# 3.0

```
public static IQueryable<TElement> AsQueryable<TElement> (
    IEnumerable<TElement> source
)
```

#### VB 9.0

```
Public Shared Function AsQueryable(Of TElement) ( _
    source As IEnumerable(Of TElement) _
) As IQueryable(Of TElement)
```

This overload wraps a nongeneric `IEnumerable` type with a loosely typed `IQueryable`:

#### C# 3.0

```
public static IQueryable AsQueryable (
    IEnumerable source
)
```

## Part II: Introducing Language Integrated Query

---

### VB 9.0

```
Public Shared Function AsQueryable ( _  
    source As IEnumerable _  
) As IQueryable
```

Replacing the verb “converts” with “wraps” is justified because `IQueryable<T>` implements `IEnumerable<T>`, `IQueryable`, and `IEnumerable`, and `IQueryable` implements `IEnumerable`.

`IQueryable` types use a second set of SQOs that substitute expression tree arguments for `IEnumerable` types’ lambda functions. Alternatively, `IQueryable` types accept custom expression trees as arguments; a custom expression tree can translate the LINQ query into another DSQL, such as SQL for LINQ to SQL. `IQueryProvider` types manage the translation, execute the DSQL query, and return the result as an `IQueryable` or `IOrderedQueryable`. Substituting `IQueryable` for `IEnumerable` allows dynamic selection of local or remote (server) processing of the same query.

The `AsQueryable` SQO supports deferred query execution by an iterator and does not force immediate execution.

The following examples execute the `query21` against local in-memory objects or an SQL Server Northwind instance, depending on the value of `runLocal` or `blnRunLocal`, assuming that the `dcNorthwind` `DataContext` is in scope:

### C# 3.0

```
IQueryable<Customer> Customers = null;  
  
bool runLocal = true;  
if (runLocal)  
    Customers = CustomerList.AsQueryable();  
else {  
    DataContext dcNorthwind = new dcNorthwind();  
    Customers = dcNorthwind;  
}  
  
var query21 = from c in Customers  
    where c.Country == "USA"  
    orderby c.CompanyName  
    select new { c.CustomerID, c.CompanyName };  
  
foreach (var c in query21)  
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {0}\r\n",  
        c.CustomerID, c.CompanyName));
```

### VB 9.0

```
Dim Customers As IQueryable(Of Customer) = Nothing  
  
Dim runLocal As Boolean = True  
If runLocal Then  
    Customers = CustomerList.AsQueryable()  
Else
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

```
DataContext dcNorthwind = new dcNorthwind();
Customers = dcNorthwind;
End If

Dim Query21 = From c In Customers _
    Where c.Country = "USA" _
    Order By c.CompanyName _
    Select c.CustomerID, c.CompanyName

For Each c In Query21
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}" & vbCrLf, _
        c.CustomerID, c.CompanyName))
Next c
```

## Cast

The default Cast SQO takes an untyped, nongeneric collection that implements `IEnumerable` and casts it to the specified generic type, if possible:

### C# 3.0

```
public static IEnumerable<TResult> Cast<TResult> (
    IEnumerable source
)
```

### VB 9.0

```
Public Shared Function Cast(Of TResult) ( _
    source As IEnumerable _
) As IEnumerable(Of TResult)
```

This Cast overload takes an untyped, nongeneric collection that implements `IQueryable` and casts it to the specified type, if possible:

### C# 3.0

```
public static IQueryable<TResult> Cast<TResult> (
    IQueryable source
)
```

### VB 9.0

```
Public Shared Function Cast(Of TResult) ( _
    source As IQueryable _
) As IQueryable(Of TResult)
```

For example, the `ArrayList` collection implements `IEnumerable` but not `IEnumerable<T>`. Cast lets you convert an `ArrayList` of compatible objects to `IEnumerable<TSource>`, in this case

## Part II: Introducing Language Integrated Query

---

`IEnumerable<Customer>`. The following examples generate an `ArrayList` containing `Customer` elements and cast the `ArrayList` to an `IEnumerable<Customer>` object:

### C# 3.0

```
ArrayList CustomersArray = new ArrayList();
foreach (var c in CustomerList)
    CustomersArray.Add(c);

var query22 = from c in CustomersArray.Cast<Customer>()
    where c.Country == "USA"
    orderby c.CompanyName
    select new { c.CustomerID, c.CompanyName };

foreach (var c in query22)
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {0}\r\n",
        c.CustomerID, c.CompanyName));
```

### VB 9.0

```
Dim CustomersArray As New ArrayList()
For Each c In CustomerList
    CustomersArray.Add(c)
Next c

Dim Query22 = From c In CustomersArray.Cast(Of Customer)() _
    Where c.Country = "USA" _
    Order By c.CompanyName _
    Select c.CustomerID, c.CompanyName

For Each c In Query22
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {0}" + vbCrLf,
        c.CustomerID, c.CompanyName))
Next c
```

The `Cast` SQO supports deferred query execution by an iterator and does not force immediate execution.

## OfType

The default `OfType<T>` operator filters a collection of multiple object types that implements `IEnumerable` to return only the type of object specified by `OfType<T>`'s argument:

### C# 3.0

```
public static IEnumerable<TResult> OfType<TResult> (
    IEnumerable source
)
```

### VB 9.0

```
Public Shared Function OfType(Of TResult) ( _
    source As IEnumerable _
) As IEnumerable(Of TResult)
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

This overload of the `OfType<T>` operator filters a collection of multiple object types that implements `IQueryable` to return only the type of object specified by `OfType<T>`'s argument:

### C# 3.0

```
public static IQueryable<TResult> OfType<TResult> (
    IQueryable source
)
```

### VB 9.0

```
Public Shared Function OfType(Of TResult) ( _
    source As IQueryable _ 
) As IQueryable(Of TResult)
```

These two examples use the `OfType` SQO to filter an `ArrayList` of `Product` and `Customer` objects to return only `Customer` objects to the `Cast` SQO:

### C# 3.0

```
ArrayList MixedArray = new ArrayList();
foreach (var p in ProductList)
    MixedArray.Add(p);
    foreach (var c in CustomerList)
        MixedArray.Add(c);

var query23 = from c in MixedArray.OfType<Customer>().Cast<Customer>()
    where c.Country == "USA"
    orderby c.CompanyName
    select new { c.CustomerID, c.CompanyName };

foreach (var c in query23)
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {0}\r\n",
        c.CustomerID, c.CompanyName));
```

The `OfType<Customer>` method must precede the `Cast<Customer>` method. Reversing the invocation order results in a runtime type mismatch exception.

### VB 9.0

```
Dim MixedArray As New ArrayList()
For Each p In ProductList
    MixedArray.Add(p)
Next p
For Each c In CustomerList
    MixedArray.Add(c)
Next c

Dim Query23 = From c In CustomersArray.OfType(Of Customer)().Cast(Of Customer)() _
    Where c.Country = "USA" _
    Order By c.CompanyName _
    Select c.CustomerID, c.CompanyName

For Each c In Query23
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {0}" + vbCrLf,
        c.CustomerID, c.CompanyName))
Next c
```

## Part II: Introducing Language Integrated Query

---

The `OfType` SQO supports deferred query execution by an iterator and does not force immediate execution.

### To . . . Operators

The four `To . . .` SQOs — `ToArray`, `ToList`, `To Dictionary`, and `ToLookup` — convert the `IEnumerable<T>` output of a LINQ query to an `Array`, `List<T>`, `Dictionary<T, K>`, or `Lookup<T, K>` type, respectively. Chaining any `To . . .()` method to a LINQ query forces immediate query execution.

The primary use of the `To . . .()` operators is to store the entire output sequence in memory for analysis, reuse, or both.

#### ToArray

The `ToArray` SQO creates an `Array()` object from an `IEnumerable<TSource>` sequence and forces immediate query execution:

##### C# 3.0

```
public static TSource[] ToArray<TSource> (
    IEnumerable<TSource> source
)
```

##### VB 9.0

```
Public Shared Function ToArray(Of TSource) ( _
    source As IEnumerable(Of TSource) _ 
) As TSource()
```

Following are C# 3.0 and VB 9.0 examples of creating an `Array()` object with `ToArray()`:

##### C# 3.0

```
Array() query24a =
(from c in CustomerList
 where c.Country == "USA"
 orderby c.CustomerID
 select c).ToArray();
```

##### VB 9.0

```
Dim Query24A As Array = _
(From c In CustomerList _
Where c.Country = "USA" _
Order By c.CustomerID _
Select c).ToArray()
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

### ToList

The `ToList` SQO creates a generic `List<TSource>` object from an `IEnumerable<TSource>` sequence and forces immediate query execution:

#### C# 3.0

```
public static List<TSource> ToList<TSource> (
    IEnumerable<TSource> source
)
```

#### VB 9.0

```
Public Shared Function ToList(Of TSource) ( _
    source As IEnumerable(Of TSource) _ 
) As List(Of TSource)
```

Here are C# 3.0 and VB 9.0 query expressions for creating a `List<Customer>` object with `ToList()`:

#### C# 3.0

```
List<Customer> query24b =
    (from c in CustomerList
     where c.Country == "USA"
     orderby c.CustomerID
     select c).ToList();
```

#### VB 9.0

```
Dim Query24B As List(Of Customer) = _
    (From c In CustomerList
     Where c.Country = "USA" _
     Order By c.CustomerID _
     Select c).ToList()
```

### To Dictionary

The `ToDictionary` SQO creates a generic `Dictionary<TKey, TElement>` collection from an `IEnumerable<TSource>` sequence in accordance with a specified key selector function and forces immediate query execution:

#### C# 3.0

```
public static Dictionary<TKey, TSource> ToDictionary<TSource, TKey> (
    IEnumerable<TSource> source,
    Func<TSource, TKey> keySelector
)
```

#### VB 9.0

```
Public Shared Function ToDictionary(Of TSource, TKey) ( _
    source As IEnumerable(Of TSource), _
    keySelector As Func(Of TSource, TKey) _
) As Dictionary(Of TKey, TSource)
```

## Part II: Introducing Language Integrated Query

---

This overload creates a Dictionary<TKey, TElement> collection from an IEnumerable<T> in accordance with specified key selector and element selector functions and forces immediate query execution:

### C# 3.0

```
public static Dictionary<TKey, TElement> ToDictionary<TSource, TKey, TElement> (  
    IEnumerable<TSource> source,  
    Func<TSource, TKey> keySelector,  
    Func<TSource, TElement> elementSelector  
)
```

### VB 9.0

```
Public Shared Function ToDictionary(Of TSource, TKey, TElement) ( _  
    source As IEnumerable(Of TSource), _  
    keySelector As Func(Of TSource, TKey), _  
    elementSelector As Func(Of TSource, TElement) _  
) As Dictionary(Of TKey, TElement)
```

The preceding two implementations also have IEqualityComparer<TKey> overloads to specify a custom comparator.

These are C# 3.0 and VB 9.0 examples of creating a Dictionary<TKey, TElement> object with ToDictionary():

### C# 3.0

```
Dictionary<string, Customer> query24c =  
    (from c in CustomerList  
     where c.Country == "USA"  
     orderby c.CustomerID  
     select c).ToDictionary(k => k.CustomerID);  
  
foreach (var c in query24c)  
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}\r\n",  
        c.Key, c.Value.CompanyName));
```

### VB 9.0

```
Dim Query24C As Dictionary(Of String, Customer) = _  
    (From c In CustomerList _  
     Where c.Country = "USA" _  
     Order By c.CustomerID _  
     Select c).ToDictionary(Function(k) k.CustomerID)  
  
For Each c In Query24C  
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {0}" + vbCrLf, _  
        c.Key, c.Value.CompanyName))  
Next c
```

Iterator code is included in the preceding samples because it departs from conventional syntax for IEnumerable<T> types.

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

### ToLookup

A `Lookup<TKey, TElement>` collection is a strongly typed collection of keys that each map to one or more values; the collection represents a read-only many-to-many association. A `Lookup` collection corresponds to a read-only Access 2007 `Lookup` field data type or a SharePoint List's `Lookup` column type. There's no public constructor for the `Lookup` collection, so `ToLookup()` is the only way to create one. `Lookup` objects are immutable, which means that you cannot add or remove elements or keys after you create the instance with `ToLookup()`.

Despite its name, the `ToLookup` SQO creates an `ILookup<TKey, TElement>` interface definition from an `IEnumerable<TSource>` in accordance with a specified key selector function:

#### C# 3.0

```
public static ILookup<TKey, TSource> ToLookup<TSource, TKey> (
    IEnumerable<TSource> source,
    Func<TSource, TKey> keySelector
)
```

#### VB 9.0

```
Public Shared Function ToLookup(Of TSource, TKey) ( _
    source As IEnumerable(Of TSource), _
    keySelector As Func(Of TSource, TKey) _
) As ILookup(Of TKey, TSource)
```

This overload creates an `ILookup<TKey, TElement>` interface definition from an `IEnumerable<T>` in accordance with specified key selector and element selector functions:

#### C# 3.0

```
public static ILookup<TKey, TElement> ToLookup<TSource, TKey, TElement> (
    IEnumerable<TSource> source,
    Func<TSource, TKey> keySelector,
    Func<TSource, TElement> elementSelector
)
```

#### VB 9.0

```
Public Shared Function ToLookup(Of TSource, TKey, TElement) ( _
    source As IEnumerable(Of TSource), _
    keySelector As Func(Of TSource, TKey), _
    elementSelector As Func(Of TSource, TElement) _
) As ILookup(Of TKey, TElement)
```

*Online help states that `ILookup<TKey, TElement>` overload “[d]efines an indexer, size property and Boolean search method for data structures that map keys to `IEnumerable<(Of <T>)>` sequences of values.” There is no online help topic for `ILookup<TKey, TSource>`.*

The preceding two implementations also have `IEqualityComparer<TKey>` overloads to specify a custom comparator.

## Part II: Introducing Language Integrated Query

---

These are C# 3.0 and VB 9.0 query expressions for creating a `IEnumerable<T>` object with `ToLookup()`:

### C# 3.0

```
ILookup<string, Customer> query24d =  
    (from c in CustomerList  
     where c.Country == "USA"  
     orderby c.CustomerID  
     select c).ToLookup(k => k.CustomerID);  
  
query24d = (Lookup<string, Customer>)query24d;  
  
foreach (var c in query24d)  
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}\r\n",  
        c.Key, c.First().CompanyName));
```

### VB 9.0

```
Dim Query24D As Lookup(Of String, Customer) = _  
    (From c In CustomerList _  
     Where c.Country = "USA" _  
     Order By c.CustomerID _  
     Select c).ToLookup(Function(k) k.CustomerID)  
  
For Each c In Query24D  
    sbResult.Append(String.Format("CustomerID = {0}, CompanyName = {1}" & vbCrLf, _  
        c.Key, c.First().CompanyName))  
Next c
```

You can cast `ILookup` to `Lookup` because the underlying implementation of the `ILookup` interface is `Lookup`. C# requires an explicit cast from `ILookup` to `Lookup`; VB performs an implicit cast. Iterator code is included in the preceding samples because the code departs from conventional syntax for `IEnumerable<T>` types.

## Equality Operator: SequenceEqual

The `SequenceEqual` SQO compares two `IEnumerable<T>` sequences. If all elements are equal in value, in the same order, and have the same count, the operator returns `True`; if not, it returns `False`. The `SequenceEqual()` operator causes immediate execution of both queries involved in creating the sequences.

Here are the method signatures:

### C# 3.0

```
public static bool SequenceEqual<TSource> (  
    IEnumerable<TSource> first,  
    IEnumerable<TSource> second  
)
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

### VB 9.0

```
Public Shared Function SequenceEqual(Of TSource) ( _  
    first As IEnumerable(Of TSource), _  
    second As IEnumerable(Of TSource) _  
) As Boolean
```

Following are simple tests of the `SequenceEqual()` operator:

### C# 3.0

```
var query25a = from c in CustomerList  
                where c.Country == "USA"  
                orderby c.CustomerID ascending  
                select c;  
  
var query25b = from c in CustomerList  
                where c.Country == "USA"  
                orderby c.CustomerID descending  
                select c;  
  
var query25c = from c in CustomerList  
                where c.Country == "USA"  
                orderby c.CustomerID ascending  
                select c;  
  
sbResult.Append("query25b is equal to query 25a? = " +  
    query25a.SequenceEqual(query25b).ToString() + "\r\n");  
sbResult.Append("query25c is equal to query 25a? = " +  
    query25a.SequenceEqual(query25c).ToString() + "\r\n");
```

### VB 9.0

```
Dim Query25A = From c In CustomerList _  
                 Where c.Country = "USA" _  
                 Order By c.CustomerID Ascending  
  
Dim Query25B = From c In CustomerList _  
                 Where c.Country = "USA" _  
                 Order By c.CustomerID Descending  
  
Dim Query25C = From c In CustomerList _  
                 Where c.Country = "USA" _  
                 Order By c.CustomerID Ascending  
  
sbResult.Append("Query25B is equal to Query 25A? = " &_  
    Query25A.SequenceEqual(Query25B).ToString() & vbCrLf)  
sbResult.Append("Query25C is equal to Query 25A? = " &  
    Query25A.SequenceEqual(Query25C).ToString() & vbCrLf)
```

The first answer is “False” because the order of `Query25B` doesn’t match that of `Query25A`; the second answer is “True” because the two sequences are clones.

# Element operators

Element SQOs return a single, specific element from an `IEnumerable<T>` sequence. A common use of element operators is to return element(s) from a sequence whose source is a related (associated) entity. If the request element isn't present, all ... Default operators return an empty object instance but don't throw an exception. The `First`, `Last`, `Single` and `ElementAt` SQOs throw an exception if the element is missing.

Element operators return singletons, so they cause immediate execution of the query that produces the sequence.

The `First()`, `Last()`, `Single()`, `FirstOrDefault()`, `LastOrDefault()`, and `SingleOrDefault()` operators share the following default method signatures, which differ only by method name:

### C# 3.0

```
public static TSource OperatorNameOrDefault<TSource> (
    IEnumerable<TSource> source
```

### VB 9.0

```
Visual Basic (Declaration)
Public Shared Function OperatorNameOrDefault(Of TSource) ( _
    source As IEnumerable(Of TSource) _ 
) As TSource
```

The following overloads test only elements that meet the condition expressed by the predicate function:

### C# 3.0

```
public static TSource OperatorNameOrDefault<TSource> (
    IEnumerable<TSource> source
    Func<TSource, bool> predicate
```

### VB 9.0

```
Visual Basic (Declaration)
Public Shared Function OperatorNameOrDefault(Of TSource) ( _
    source As IEnumerable(Of TSource) _ 
    predicate As Func(Of TSource, Boolean) _ 
) As TSource
```

The `QueryOperatorsCS.sln` and `QueryOperatorsVB.sln` projects contain `query26` and `Query 26` sample queries that demonstrate the `First()`, `Last()`, `ElementAtOrDefault()` and `SingleOrDefault()` operators.

## ***First, FirstOrDefault***

The `First()` and `FirstOrDefault()` operators return the initial element of their source sequence, if the sequence contains one or more elements. If not, the `FirstOrDefault()` operator returns a null instance for reference types and `Default<T>` for value types; `First()` throws an exception.

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

The following snippets return the element for the first U.S. customer alphabetically:

### C# 3.0

```
var query26a = (from c in CustomerList
                 where c.Country == "USA"
                 orderby c.CustomerID ascending
                 select c).First();
```

### VB 9.0

```
Dim Query26A = (From c In CustomerList _
                 Where c.Country = "USA" _
                 Order By c.CustomerID Ascending _
                 Select c).First()
```

The earlier “ToLookup” section about the ToLookup SQO is a typical application for the `First()` element operator.

## Last, LastOrDefault

The `Last()` and `LastOrDefault()` operators return the final element of their source sequence, if the sequence contains one or more elements. If not, the `LastOrDefault()` operator returns a null instance and `Default<T>` for value types; `Last()` throws an exception.

These snippets use a predicate function return the element for the last Brazilian customer alphabetically:

### C# 3.0

```
var query26b = (from c in CustomerList
                 orderby c.CustomerID ascending
                 select c).Last(c => c.Country == "Brazil");
```

### VB 9.0

```
Dim Query26B = (From c In CustomerList _
                 Order By c.CustomerID Ascending _
                 Select c).Last(Function(c) c.Country = "Brazil")
```

## Single, SingleOrDefault

The `Single()` and `SingleOrDefault()` operators return the final element of their source sequence, if the sequence contains one and only one element. If the sequence is empty, `Single()` throws an exception and `SingleOrDefault()` returns a null instance for reference types and `Default<T>` for value types. If the sequence contains more than one element, either operator throws an exception.

These snippets use a predicate function to return an empty element because there are no Nigerian customers:

### C# 3.0

```
var query26c = (from c in CustomerList
                 orderby c.CustomerID ascending
                 select c).SingleOrDefault(c => c.Country == "Nigeria");
```

## Part II: Introducing Language Integrated Query

---

### VB 9.0

```
Dim Query26C = (From c In CustomerList _
                 Order By c.CustomerID Ascending _
                 Select c).SingleOrDefault(Function(c) c.Country = "Nigeria")
```

### DefaultIfEmpty

The `DefaultIfEmpty()` operator returns the defined default instance as a singleton if the specified source element or range is missing or empty.

The earlier `GroupJoin` section's example of creating the LINQ equivalent of an SQL left outer join uses the `DefaultIfEmpty()` operator to ensure that each outer item appears even if it has no corresponding inner items(s). This example includes the code required to create the default instance.

### ElementAt, ElementAtOrDefault

The `ElementAt()` and `ElementAtOrDefault()` operators return the element whose location in the sequence is specified by the index value, if the element is present. If missing, the `ElementAtOrDefault()` operator returns the default instance; `ElementAt()` throws an exception.

Here are the two method signatures:

### C# 3.0

```
public static TSource ElementAt[OrDefault]<TSource> (
    IEnumerable<TSource> source,
    int index
)
```

### VB 9.0

```
Visual Basic (Declaration)
Public Shared Function ElementAt[OrDefault](Of TSource) ( _
    source As IEnumerable(Of TSource), _
    index As Integer _
) As TSource
```

The following snippets return the element for the fifth U.S. customer alphabetically:

### C# 3.0

```
var query26d = (from c in CustomerList
                 where c.Country == "USA"
                 orderby c.CustomerID ascending
                 select c).ElementAtOrDefault(5);
```

### VB 9.0

```
Dim Query26D = (From c In CustomerList _
                 Where c.Country = "USA" _
                 Order By c.CustomerID Ascending _
                 Select c).ElementAtOrDefault(5)
```

# Generation Operators

Generation operators return one of three specific types of sequence. VS 2008's online help offers contrived suggestions for the use of the sequences. Nontrivial uses don't come readily to mind.

Method signatures aren't interesting, so sample code to generate the sequences is provided instead.

## Range

`Range()` generates a collection that contains a sequence of numbers. The following snippets create 10 elements containing values of 1 to 10:

### C# 3.0

```
IEnumerable<int> oneToTen = Enumerable.Range(1, 10);
```

### VB 9.0

```
Dim OneToTen As IEnumerable(Of Integer) = Sequence.Range(1, 10)
```

## Repeat

`Repeat()` generates a collection that contains one repeated value. The following snippets write a chalkboard phrase 100 times:

### C# 3.0

```
IEnumerable<string> bartSimpson =
    Sequence.Repeat("I will not bribe Principal Skinner.", 100);
```

### VB 9.0

```
Dim BartSimpson As IEnumerable(Of String) =
    Sequence.Repeat("I will not bribe Principal Skinner.", 100)
```

## Empty

The `Empty()` operator returns an empty sequence of the specified type. The following snippets create an empty `IEnumerable<Customer>` sequence:

### C# 3.0

```
IEnumerable<Customer> emptyCustomer = Enumerable.Empty<Customer>();
```

### VB 9.0

```
Dim emptyCustomer As IEnumerable(Of Customer) = Enumerable.Empty(Of Customer)()
```

# Quantifier Operators

Quantifier operators return a `bool/Boolean` value to indicate whether some or all of the elements in a sequence satisfy a specified condition.

Quantifier operators return singletons, so their use causes immediate query execution.

## Any

The `Any()` operator returns `True` if any element in a sequence satisfies a condition.

### C# 3.0

```
public static bool All<TSource> (
    IEnumerable<TSource> source,
    Func<TSource, bool> predicate
)
```

### VB 9.0

```
Public Shared Function All(Of TSource) ( _
    source As IEnumerable(Of TSource), _
    predicate As Func(Of TSource, Boolean) _
) As Boolean
```

The following statements return `True` if any `Order` object has a `Freight` value greater than US\$100:

### C# 3.0

```
bool anyFreightOver100 = OrderList.Any(c => c.Freight > 100.00M);
```

### VB 9.0

```
Dim AnyFreightOver100 = OrderList.Any(Function(c) c.Freight > 100D)
```

## All

The `All()` operator returns `True` if all elements in a sequence satisfy a condition.

### C# 3.0

```
public static bool All<TSource> (
    IEnumerable<TSource> source,
    Func<TSource, bool> predicate
)
```

### VB 9.0

```
Public Shared Function All(Of TSource) ( _
    source As IEnumerable(Of TSource), _
    predicate As Func(Of TSource, Boolean) _
) As Boolean
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

The following statements return `True` if all `Order` objects have a `Freight` value greater than or equal to US\$5:

### C# 3.0

```
bool allFreight5OrMore = OrderList.All(c => c.Freight >= 5.00M);
```

### VB 9.0

```
Dim AllFreight5OrMore = OrderList.All(Function(c) c.Freight >= 5D)
```

## Contains

The `Contains()` operator returns `True` if a sequence contains a specified element.

### C# 3.0

```
public static bool Contains<TSource> (
    IEnumerable<TSource> source,
    TSource value
)
```

### VB 9.0

```
Public Shared Function Contains(Of TSource) ( _
    source As IEnumerable(Of TSource), _
    value As TSource _
) As Boolean
```

The following statements return `True` if the `CustomerList` collection contains the last `Customer` element:

### C# 3.0

```
Customer testCustomer = CustomerList.Last();
```

### VB 9.0

```
bool containsLastCustomer = CustomerList.Contains(testCustomer);
```

## Aggregate Operators

The default Aggregate SQO implementations, with the exception of `Aggregate`, emulate the corresponding SQL aggregate functions. Aggregate operators return singletons so they cause immediate query execution.

Chapter 4 contains code examples for aggregating values in groups of elements generated by the `GroupJoin` and `GroupBy` operators.

## Part II: Introducing Language Integrated Query

---

### Count and LongCount

The default `Count()` and `LongCount()` implementations accept an `IEnumerable<TSource>` type and return an integer: `int32` or `int64` for C#'s `dataType` and `Integer` or `Long` for VB's `DataType`:

#### C# 3.0

```
public static dataType aggregateName<TSource> (
    IEnumerable<TSource> source
)
```

#### VB 9.0

```
Public Shared Function AggregateName(Of TSource) ( _
    source As IEnumerable(Of TSource) _ 
) As DataType
```

Overloads add a predicate function so that aggregation occurs only for elements that satisfy a condition:

#### C# 3.0

```
public static dataType aggregateName<TSource> (
    IEnumerable<TSource> source,
    Func<TSource, bool> predicate
)
```

#### VB 9.0

```
Public Shared Function AggregateName(Of TSource) ( _
    source As IEnumerable(Of TSource), _
    predicate As Func(Of TSource, Boolean) _ 
) As DataType
```

The following statements return the number of orders shipped to the U.S.:

#### C# 3.0

```
int countUsOrders = OrderList.Count(c => c.ShipCountry == "USA");
```

#### VB 9.0

```
Dim CountUsOrders As Long = OrderList.LongCount(Function(c) c.ShipCountry = "USA")
```

*Using `Count()` with VB 9.0 throws an exception. Although this appears to be a bug, the Microsoft Connect forum closed the bug report with "By Design."*

### **Min, Max, Sum, and Average**

Following are the default and Nullable overload method signatures of the `Min()` aggregate operator for the `decimal/Decimal` data type with a predicate to specify the name of the numeric property value to aggregate:

```
public static decimal Min (
    IEnumerable<decimal> source
    Func<TSource, decimal> selector
)

public static Nullable<decimal> Min (
    IEnumerable<Nullable<decimal>> source
    Func<TSource, decimal> selector
)
Public Shared Function Min ( _
    source As IEnumerable(Of Decimal) _
    selector As Func(Of TSource, Decimal) _
) As Decimal

Public Shared Function Min ( _
    source As IEnumerable(Of Nullable(Of Decimal)) _
    selector As Func(Of TSource, Decimal) _
) As Nullable(Of Decimal)
```

Signatures for the `Max()`, `Sum()`, and `Avg()` operators and the `int32/Integer`, `int64/Long`, `double/Double`, `single/Single` data types are similar to the preceding. There are many other overloads for aggregating numeric property values, but the preceding are the most popular.

The following statements return the minimum, maximum, total, and average freight charge for all orders in an `OrderList` instance:

#### **C# 3.0**

```
decimal? minFreightCharge = OrderList.Min(c => c.Freight);

decimal? maxFreightCharge = OrderList.Max(c => c.Freight);

decimal? sumFreightCharge = OrderList.Sum(c => c.Freight);

decimal? avgFreightCharge = OrderList.Average(c => c.Freight);
```

#### **VB 9.0**

```
Dim MinFreightCharge As Decimal? = OrderList.Min(Function(c) c.Freight)

Dim MaxFreightCharge As Decimal? = OrderList.Max(Function(c) c.Freight)

Dim SumFreightCharge As Decimal? = OrderList.Sum(Function(c) c.Freight)

Dim AvgFreightCharge As Decimal? = OrderList.Average(Function(c) c.Freight)
```

## Part II: Introducing Language Integrated Query

---

### Aggregate

The Aggregate SQO lets you define a custom aggregation operation that performs a calculation over a sequence of values and returns the result.

Following is `Aggregate()` operator's default method signature:

#### C# 3.0

```
public static TSource Aggregate<TSource> (
    IEnumerable<TSource> source,
    Func<TSource1, TSource2, TSource3> func
)
```

#### VB 9.0

```
Public Shared Function Aggregate(Of TSource) ( _
    source As IEnumerable(Of TSource), _
    func As Func(Of TSource1, TSource2, TSource3) _
) As TSource
```

The `Aggregate()` operator calls the `func` function once for each `source` element. The `TSource1` argument holds the current value (often called the *seed*), `TSource2` holds the source value and `TSource3` holds the new computed value. `TSource3` replaces `TSource1` after each `func` invocation. The `Aggregate()` operator returns the final value of `TSource3` to the caller.

The following overload enables supplying a non-default `seed` value that can have a different data type than `source`:

#### C# 3.0

```
public static TAccumulate Aggregate<TSource, TAccumulate> (
    IEnumerable<TSource> source,
    TAccumulate seed,
    Func<TAccumulate, TSource, TAccumulate> func
)
```

#### VB 9.0

```
Public Shared Function Aggregate(Of TSource, TAccumulate) ( _
    source As IEnumerable(Of TSource), _
    seed As TAccumulate, _
    func As Func(Of TAccumulate, TSource, TAccumulate) _
) As TAccumulate
```

This overload adds the capability to provide a custom `resultSelector` function to supply the final result:

#### C# 3.0

```
public static TResult Aggregate<TSource, TAccumulate, TResult> (
    IEnumerable<TSource> source,
    TAccumulate seed,
    Func<TAccumulate, TSource, TAccumulate> func,
    Func<TAccumulate, TResult> resultSelector
)
```

## Chapter 3: Executing LINQ Query Expressions with LINQ to Objects

---

### VB 9.0

```
Public Shared Function Aggregate(Of TSource, TAccumulate, TResult) ( _  
    source As IEnumerable(Of TSource), _  
    seed As TAccumulate, _  
    func As Func(Of TAccumulate, TSource, TAccumulate), _  
    resultSelector As Func(Of TAccumulate, TResult) _  
) As TResult
```

The following expressions use the `Aggregate()` operator's default implementation to return the same values as the preceding section's `Min`, `Max`, `Sum`, and `Average` SQO statements:

### C# 3.0

```
decimal? minFreightChargeAgg = OrderList.Select(o => o.Freight)  
    .Aggregate((seed, f) => f <= seed ? f : seed);  
  
decimal? maxFreightChargeAgg = OrderList.Select(o => o.Freight)  
    .Aggregate((seed, f) => f > seed ? f : seed);  
  
decimal? sumFreightChargeAgg = OrderList.Select(o => o.Freight)  
    .Aggregate((seed, f) => seed + f);  
  
decimal? avgFreightChargeAgg = sumFreightChargeAgg/OrderList.Count();
```

### VB 9.0

```
Dim MinFreightChargeAgg As Decimal? = OrderList.Select(Function(o) o.Freight) _  
    .Aggregate(Function(seed, i) If(i <= seed, i, seed))  
Dim MaxFreightChargeAgg As Decimal? = OrderList.Select(Function(o) o.Freight) _  
    .Aggregate(Function(seed, i) If(i > seed, i, seed))  
Dim SumFreightChargeAgg As Decimal? = OrderList.Select(Function(o) o.Freight) _  
    .Aggregate(Function(seed, i) seed + i)  
Dim AvgFreightChargeAgg As Decimal? = SumFreightChargeAgg / OrderList.Count()
```

*Notice the use of VB 9.0's new ternary If operator in the Min ... and Max ... implementations. This operator is similar to the IIf() function, but short-circuits compound expressions.*

Most `Aggregate()` code examples use trivial numeric or string arrays as data sources. The preceding examples have the advantage of verifiability against the built-in versions.

## Summary

This chapter introduces you to the LINQ Standard Query Operators, query expressions, and keywords that form the foundation for all Microsoft and third-party LINQ implementations. The SQOs are classified into groups defined by Microsoft's "The .NET Standard Query Operators" whitepaper of February 2007. The C# 3.0 and VB 9.0 versions of the LINQ Project Sample Query Explorer projects let you execute a set of more than 100 LINQ query expressions against a variety of data sources that range from simple in-memory string and numeric arrays to tables of the Northwind sample database.

## Part II: Introducing Language Integrated Query

---

The objective of this chapter is to explain and demonstrate the syntax of LINQ 1.0's 50 SQOs. To make this and succeeding chapters' examples more meaningful, the in-memory business objects that serve as the data sources for the sample queries include at least one association with a related object. The Order object has 1:1 associations with Customer, Employee, and Shipper objects and a 1:many association with Order\_Detail objects. A LINQ In-Memory Object Generation utility (LIMOG.sln) generates all class definitions, object initializers and, for C# classes, collection initializers. The C# and VB versions of the NwindObjects.sln utility populate association key values and serialize List<ObjectName> collections for hydration of the in-memory collections used by this chapter's sample QueryOperators.sln projects. You learn more about LIMOG.sln and the NwindObjects.sln projects in Chapter 4.

This chapter provides method signatures and at least one query expression or SQO method call to demonstrate how to programmatically execute all 50 SQOs. For the sake of completeness, no operators or keywords have been "omitted for brevity." There are few — if any — other technical publications that provide such a comprehensive LINQ reference in a single location and no other known sources that use real-world business objects for all examples. The C# and VB versions of QueryOperators.sln run the equivalent of 100% coverage with unit tests on all sample code in this chapter.

# 4

## Working with Advanced Query Operators and Expressions

Chapter 3 provided examples of LINQ query expressions and method call syntax for Standard Query Operators (SQOs) by group. The data sources for queries were collections of relatively simple business objects. The objects included public fields of generic types and generic collections, but these fields were empty. The LINQ query expressions used `Joins` or `GroupJoins` to emulate relationships, where appropriate.

This chapter covers more complex LINQ queries over collections of *object graphs* that emulate real-world business entities, such as customers, orders, employees, suppliers, and products, by taking advantage of the *relationships* between the entities. *Object graphs* are tree-like representations for describing collections and instances of objects and hierarchies of their dependent and related objects. An object graph provides an internally consistent view of an application's data.

Much of this chapter is devoted to explaining complex aggregate queries that use the `Count()` and `Sum()` SQOs primarily, along with the `Avg()`, `Min()`, and `Max()` operators, with or without the join and grouping operators — `Join()`, `GroupBy()`, and `GroupJoin()`. You'll also see examples of left outer joins implemented by nested `Select()` and `GroupJoin()` queries.

Other advanced topics covered by this chapter are unconventional use of the `Contains()` SQO for widening `Where` clause filters and substitutes for T-SQL's `IN()` function, compiling specific types of queries to maximize performance, and emulating object graphs generated by LINQ to SQL or LINQ to Entities implementations.

# Exploring Basic Query Syntax for Aggregate Operators

One of the most common applications for LINQ queries is generating summary reports with numerical totals and subtotals. For example, following is a summary of a single Northwind Order object generated from the OrderList and Order\_DetailList collections:

```
OrderID = 10321, CustID = ISLAT, ShipDate = 10/11/1996 12:00:00 AM, Items = 1,  
Subtotal = 144, Freight 3.43, Total = 147.43
```

The Subtotal calculation is `Quantity * UnitPrice * (1 - Discount)` summed for the order's line items and `Total = Subtotal + Freight`. The Total calculation requires temporary storage for Subtotal to eliminate redundant calculation. The following sections show examples of alternative approaches to basic aggregation syntax using the `Sum()` and `Count()` functions.

*Sample aggregate queries are contained in the \WROX\ADONET\Chapter04\CS\AdvancedQueryOperatorsCS.sln and \WROX\ADONET\Chapter04\VB\AdvancedQueryOperatorsVB.sln projects. These projects were based on Chapter 2's examples and use binary-serialized generic Lists, which require maintaining the original assembly name. Thus the executable filenames differ from the project names.*

The C# and VB AdvancedQueryOperators forms execute all query examples of this chapter in the sequence of their appearance, except where noted. Figure 4-1 shows the VB version displaying the third, fourth, and part of the fifth LINQ aggregate query example.

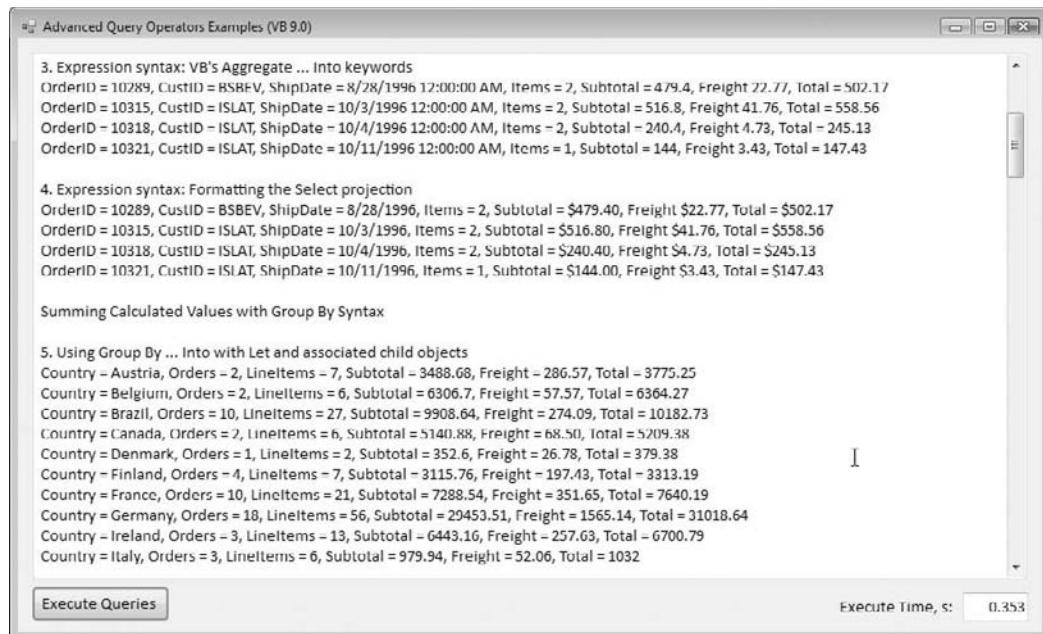


Figure 4-1

### Basic Method Call Syntax with Numerical Operators

Method call syntax doesn't support the `Let` keyword for storing temporary local variables, so the simplest approach — if you *must* use a method call — is to perform aggregation with numerical operators in a nested loop, as shown here:

#### C# 3.0

```
var usOrdersTotaled1 = OrderList.Where(o => o.ShipCountry == "UK");
foreach (var o in usOrdersTotaled1)
{
    decimal itemsTotal = 0M;
    int count = 0;

    foreach (var d in o.Order_Details)
    {
        itemsTotal += d.Quantity * d.UnitPrice * (1 - (decimal)d.Discount);
        count++;
    }
    sbResult.Append(string.Format("OrderID = {0}, CustID = {1}, ShipDate = {2}, " +
        "Items = {3}, Subtotal = {4}, Freight {5}, Total = {6}\r\n",
        o.OrderID, o.CustomerID, o.ShippedDate, count, itemsTotal,
        o.Freight, itemsTotal + o.Freight));
}
```

#### VB 9.0

```
Dim usOrdersTotaled1 = OrderList.Where(Function(o) _
    o.ShipCountry = "UK")
For Each o In usOrdersTotaled1
    Dim decItemsTotal As Decimal
    Dim intCount As Integer

    For Each d In o.Order_Details
        decItemsTotal += d.Quantity * d.UnitPrice * (1 - d.Discount)
        intCount += 1
    Next

    sbResult.Append(String.Format("OrderID = {0}, CustID = {1}, " + _
        "ShipDate = {2}, Items = {3}, Subtotal = {4}, Freight {5}, " + _
        "Total = {6}" + vbCrLf, o.OrderID, o.CustomerID, o.ShippedDate, _
        intCount, o.ItemsTotal, o.Freight, o.Subtotal))
Next
```

The preceding approach isn't efficient for queries against database data sources because the LINQ query retrieves every `Order_Detail` row instead of the aggregate values of row sets. Further, you can't invoke the `ToList()` method to deliver the data source for a `GridView` or `DataGridView` control. It's a better practice to use LINQ expression syntax, as shown in the next section, rather than method syntax for queries of this type.

## Part II: Introducing Language Integrated Query

---

Iterating the query of a `List<Order>` of the first 100 Northwind orders for orders shipped to the UK returns the first four rows shown in the later “Formatting the Query Output” section. Following is the corresponding T-SQL query:

### T-SQL

```
SELECT o.OrderID, o.CustomerID AS CustID, o.ShippedDate AS ShipDate,
       COUNT(d.OrderID) AS Items, SUM(d.Quantity * d.UnitPrice * (1 - CONVERT(money,
       d.Discount))) AS Subtotal, Freight, SUM(d.Quantity * d.UnitPrice *
       (1 - CONVERT(money, d.Discount))) + Freight AS Total
  FROM Orders AS o INNER JOIN [Order Details] AS d ON d.OrderID = o.OrderID
 WHERE o.ShipCountry = 'UK' AND o.OrderID < 10342
 GROUP BY o.OrderID, o.CustomerID, o.ShippedDate, Freight
 ORDER BY o.OrderID
```

The `CONVERT` function is required to eliminate rounding errors in the `real (Single)` `Discount` value. A 1:many association between the `Order` and `Order_Detail` objects eliminates the need for a `Join` statement in the LINQ query. SQL Server Books Online’s `GROUP BY` topic recommends that you always use the `ORDER BY` clause if your query contains `GROUP BY`.

## Expression Syntax with Let as a Temporary Local Aggregate Variable

Storing aggregate values generated by a lambda function or VB’s `Aggregate` keyword in a `let` or `Let` temporary local variable minimizes resource consumption by the sample query. The following two sections use `Let` to store the `Subtotal` value.

### Using a Lambda Function for Aggregation of Child Entity Values

Lambda functions enable and value aggregation and other computations of child entity values, as illustrated by the following examples:

#### C# 3.0

```
var usOrdersTotaled2 = from o in OrderList
                       where o.ShipCountry == "UK"
                       let itemsTotal = o.Order_Details.Sum(d => d.Quantity *
                           d.UnitPrice * (1 - (decimal)d.Discount))
                       select new { o.OrderID, o.CustomerID, o.ShippedDate,
                           o.Order_Details.Count, itemsTotal = itemsTotal,
                           o.Freight, Subtotal = itemsTotal + o.Freight };
```

#### VB 9.0

```
Dim usOrdersTotaled2 = From o In OrderList Where o.ShipCountry = "UK" _
    Let ItemsTotal = o.Order_Details.Sum(Function(d) _
        d.Quantity * d.UnitPrice * (1 - d.Discount)) _
    Select o.OrderID, o.CustomerID, o.ShippedDate, _
        o.Order_Details.Count(), ItemsTotal, o.Freight, _
        Subtotal = ItemsTotal + o.Freight
```

You can wrap the preceding query expressions in parentheses and apply the `ToList()` operator to populate the `DataSource` property of a `BindingSource` or `ObjectDataSource` component or grid controls.

*Iterators for these sample queries are similar to those shown in the preceding section.*

### Using Visual Basic's Aggregate . . . Into Keywords

Aggregate is an alternative to VB's From starting keyword that avoids use of lambda functions in aggregate expressions. Aggregate returns a scalar value by applying the numeric aggregate expression following the Into keyword to the sequence specified after the In keyword, as shown here:

#### VB 9.0

```
Dim usOrdersTotaled3 = From o In OrderList Where o.ShipCountry = "UK" _
    Let ItemsTotal = Aggregate d In o.Order_Details _
        Into Sum(d.Quantity * d.UnitPrice * (1 - d.Discount)) _
        Select o.OrderID, o.CustomerID, o.ShippedDate, _
            o.Order_Details.Count(), ItemsTotal, o.Freight, _
            Subtotal = ItemsTotal + o.Freight
```

Aggregate...Into expressions are “syntactic sugar” intended to eliminate the need for VB programmers to deal with lambda expressions. Many C# developers believe that the entire LINQ expression syntax is syntactic sugar for chained method call syntax. Syntactic sugar is defined as alternative syntax that programmers find “sweeter” or easier to use but doesn’t add functionality to the programming language.

### Formatting the Query Output

Following is the unformatted output of the three preceding queries:

```
OrderID = 10289, CustID = BSBEV, ShipDate = 8/28/1996 12:00:00 AM, Items = 2,
Subtotal = 479.4, Freight 22.77, Total = 502.17
OrderID = 10315, CustID = ISLAT, ShipDate = 10/3/1996 12:00:00 AM, Items = 2,
Subtotal = 516.8, Freight 41.76, Total = 558.56
OrderID = 10318, CustID = ISLAT, ShipDate = 10/4/1996 12:00:00 AM, Items = 2,
Subtotal = 240.4, Freight 4.73, Total = 245.13
OrderID = 10321, CustID = ISLAT, ShipDate = 10/11/1996 12:00:00 AM, Items = 1,
Subtotal = 144, Freight 3.43, Total = 147.43
```

You can format the date and currency operations by working with the Select projection values but formatting strings for reports in the iterator is preferred because the data types are preserved in the Select clause’s output. For example, the following C# iterator removes the time from ShipDate and formats Subtotal, Freight, and Total as currency:

#### C# 3.0

```
foreach (var o in usOrdersTotaled4)
{
    sbResult.Append(string.Format("OrderID = {0}, CustID = {1}, ShipDate = {2}, " +
        "Items = {3}, Subtotal = {4}, Freight {5}, Total = {6}\r\n",
        o.OrderID, o.CustomerID, ((DateTime)o.ShippedDate).ToString("d"), o.Count,
        o.itemsTotal.ToString("c"), ((Decimal)o.Freight).ToString("c"),
        ((Decimal)o.Subtotal).ToString("c")));
}
```

## Part II: Introducing Language Integrated Query

---

ShippedDate is of DateTime? (Nullable<DateTime>) type, so it must be coerced to DateTime; Freight and Subtotal are type Double and must be coerced to Decimal for currency formatting.

VB enables substituting the Format function in the iterator, which is less choosy about data types, as shown here:

### VB 9.0

```
For Each o In usOrdersTotaled4
    sbResult.Append(String.Format("OrderID = {0}, CustID = {1}, ShipDate = {2}, " + _
        "Items = {3}, Subtotal = {4}, Freight {5}, Total = {6}" + vbCrLf, _
        o.OrderID, o.CustomerID, Format(o.ShippedDate, "d"), o.Count, _
        o.ItemsTotal.ToString("c"), Format(o.Freight, "c"), Format(o.Subtotal, "c")))
```

Next

The C# version throws a “Nullable object must have a value” exception when it encounters a null ShipDate value; the VB version doesn’t. Otherwise, they both deliver the following formatted output:

```
OrderID = 10289, CustID = BSBEV, ShipDate = 8/28/1996, Items = 2,
Subtotal = $479.40, Freight $22.77, Total = $502.17
OrderID = 10315, CustID = ISLAT, ShipDate = 10/3/1996, Items = 2,
Subtotal = $516.80, Freight $41.76, Total = $558.56
OrderID = 10318, CustID = ISLAT, ShipDate = 10/4/1996, Items = 2,
Subtotal = $240.40, Freight $4.73, Total = $245.13
OrderID = 10321, CustID = ISLAT, ShipDate = 10/11/1996, Items = 1,
Subtotal = $144.00, Freight $3.43, Total = $147.43
```

## Using Group By with Aggregate Queries

The preceding aggregate query expressions return one summary row for each parent object (Order) that meets the Where clause criterion. LINQ queries against child objects with many:1 associations to their parent don’t require the GroupBy() SQO because children are grouped by the association. Returning summary data for groups of parent objects requires applying the GroupBy() operator by including a groupby member into variable] or Group By Member Into [Variable | Expression] statement in the query.

Sample queries in the following sections return 18 rows from the OrderList and Order\_Detail\_List collections of which the following are rows 1, 2, 17, and 18:

```
Country = Austria, Orders = 2, LineItems = 7, Subtotal = 3488.68,
Freight = 286.57, Total = 3775.25
Country = Belgium, Orders = 2, LineItems = 6, Subtotal = 6306.7,
Freight = 57.57, Total = 6364.27
...
Country = USA, Orders = 16, LineItems = 42, Subtotal = 29039.93,
Freight = 1470.90, Total = 30510.82
Country = Venezuela, Orders = 5, LineItems = 14, Subtotal = 6335.5,
Freight = 245.88, Total = 6581.38
```

## Chapter 4: Working with Advanced Query Operators and Expressions

---

Following is a T-SQL GROUP BY query that returns the same rows as the sample LINQ queries:

### T-SQL

```
SELECT ShipCountry AS Country, COUNT(DISTINCT o.OrderID) AS Orders,
       COUNT(d.OrderID) AS LineItems, SUM(d.Quantity * d.UnitPrice *
       (1 - CONVERT(money, d.Discount))) AS Subtotal, SUM(DISTINCT o.Freight) AS
       Freight, SUM(d.Quantity * d.UnitPrice * (1 - CONVERT(money, d.Discount))) +
       SUM(DISTINCT Freight) AS Total
  FROM Orders AS o INNER JOIN [Order Details] AS d ON d.OrderID = o.OrderID
 WHERE o.OrderID < 10342
 GROUP BY o.ShipCountry
 ORDER BY o.ShipCountry
```

*COUNT(DISTINCT o.OrderID) returns the correct count of orders, but SUM(DISTINCT o.Freight) doesn't return the correct freight costs if two or more orders in the group have the same freight charge. This qualifies the query as nondeterministic.*

## Grouping with Associated Child Objects

Real-world business objects almost always associate child objects with their parent objects by a 1:many relationship, and child objects often include references to parent objects by many:1 relationships. Associating child objects with parent objects maintains the hierarchical structure and provides explicit grouping at the lowest level of the hierarchy. If your objects have 1:many associations, the better practice is to take advantage of them when preparing summary reports. One advantage of maintaining the object hierarchy when grouping objects is to enable accurate aggregation of numeric property values of the parent object, such as Freight in the preceding T-SQL example. Both of the following queries are deterministic as to Freight calculation.

*The VB version of the query example appears first in the following sections because VB 9.0 is the language preferred by many developers for grouping with LINQ queries demonstrations.*

## Using the Aggregate Keyword and Into Expression with VB

The following example illustrates preaggregating child record (o.Order\_Details) values into local temporary variables (ItemsTotal and ItemsCount), which you sum for the grouping interval (ShipCountry).

### VB 9.0

```
Dim ordersByCountry5 = From o In OrderList _
    Let ItemsTotal = Aggregate d In o.Order_Details _
        Into Sum(d.Quantity * d.UnitPrice * (1 - d.Discount)) _
    Let ItemsCount = Aggregate l In o.Order_Details _
        Into Count() _
    Order By o.ShipCountry _
    Group By o.ShipCountry _
        Into SubTotal = Sum(ItemsTotal), Orders = Count(), _
        LineItems = Sum(ItemsCount), Freight = Sum(o.Freight), _
```

## Part II: Introducing Language Integrated Query

---

```
Total = Sum(ItemsTotal + o.Freight) _  
  
For Each o In usOrdersTotaled5  
    sbResult.Append(String.Format("Country = {0}, Orders = {1}, " + _  
        "LineItems = {2}, Subtotal = {3}, Freight = {4}, Total = {5}" + vbCrLf, _  
        o.ShipCountry, o.Orders, o.LineItems, o.SubTotal, o.Freight, o.Total))  
Next
```

VB lets you assign the equivalent of an aggregate `Select` projection list to the `Group By` instruction with the `Into` operator, which makes a `Select` list optional. C# doesn't support this construct, as demonstrated in the next section.

*You can replace the `Count()` operator with any combination of the `Count()`, `Min()`, or `Max()` operators in the same location in the expression. You also can replace `Sum(Expression)` with or add `Avg(Expression)` in the same location.*

### Using C# Group By Expression Syntax

C#'s requirement to designate a variable to hold the grouping key and sequence prevents assigning preaggregated values variables (assigned by `let`) to the `select` projection. The compiler rejects these variables in the projection because they are out of scope; the `select` expression can contain the `group.Key` value and members of the `group` sequence only. You must write lambda expressions to aggregate out-of-scope values, such as `Subtotal` and `Freight`. There's no opportunity for temporary storage, so you must repeat the `Subtotal` and `Freight` aggregations to obtain the `Total` value.

These restrictions complicate the C# `select` expression greatly, as shown here:

#### C# 3.0

```
var ordersByCountry5 = from o in OrderList  
    /* let itemsTotal = o.Order_Details.Sum(d => d.Quantity *  
       d.UnitPrice * (1 - (decimal)d.Discount))  
    let itemsCount = o.Order_Details.Count() */  
    orderby o.ShipCountry  
    group o by o.ShipCountry into g  
    select new { Country = g.Key, Orders = g.Count(),  
        LineItems = g.Sum(d => d.Order_Details.Count()),  
        Subtotal = g.Sum(d => d.Order_Details.Sum(l =>  
            l.Quantity * l.UnitPrice * (1 - (decimal)l.Discount))),  
        Freight = g.Sum(o => o.Freight),  
        Total = g.Sum(d => d.Order_Details.Sum(l =>  
            l.Quantity * l.UnitPrice * (1 - (decimal)l.Discount)))  
            + g.Sum(o => o.Freight) };  
  
foreach (var o in ordersByCountry5)  
{  
    sbResult.Append(String.Format("Country = {0}, Orders = {1}, LineItems = {2},  
        Subtotal = {3}, Freight = {4}, Total = {5}\r\n",  
        o.Country, o.Orders, o.LineItems, o.Subtotal, o.Freight, o.Total));  
}
```

I believe you'll agree that the VB versions of the preceding LINQ grouping expressions are much more readable than their C# counterparts, and even more so than chained method calls in either language.

### Grouping with Joined Child Objects

If your related entities don't have their associations populated, you must create a join between the entities with the `join...in...on...equals...` expression or its mixed-case VB counterpart. Joins generate a flat view of the object hierarchy. For example, the following C# query returns the same rows as the preceding two examples:

#### C# 3.0

```
var ordersByCountry6 = from o in OrderList
                      join d in Order_DetailList on o.OrderID equals d.OrderID
                      orderby o.ShipCountry
                      group d by o.ShipCountry into g
                      select new
                      {
                          Country = g.Key,
                          LineItems = g.Count(),
                          Subtotal = g.Sum(d => d.Quantity * d.UnitPrice *
                                         (1 - (decimal)d.Discount))
                      };
}

foreach (var o in ordersByCountry6)
{
    var ord = o; // Suppress iteration variable warning
    var Freight = OrderList.Where(n => n.ShipCountry ==
        ord.Country).Sum(f => f.Freight);
    sbResult.Append(String.Format("Country = {0}, Orders = {1}, LineItems = {2},
        Subtotal = {3}, Freight = {4}, Total = {5}\r\n",
        ord.Country, OrderList.Where(n => n.ShipCountry == ord.Country).Count(),
        ord.LineItems, ord.Subtotal, Freight, ord.Subtotal + Freight));
}
```

*If you use the `o` iteration variable instead of the `ord` inferred type variable, such as in `o.Country`, you receive the following warning from the C# or VB compiler: "Using the iteration variable in a lambda expression may have unexpected results. Instead, create a local variable within the loop and assign it the value of the iteration variable." In this case, unexpected results don't occur because the iteration variable is deterministic.*

The C# sample comes first because the `group d by o.ShipCountry into g` expression is more explicit about its source sequence (`d`) than VB's conventional `Group By o.ShipCountry Into Subtotal = ...` expression and the lambda expression syntax in the iterator is more concise. Here's the VB version:

#### VB 9.0

```
Dim ordersByCountry6 = From o In OrderList _
                      Join d In Order_DetailList On d.OrderID Equals o.OrderID _
                      Order By o.ShipCountry _
                      Group By o.ShipCountry _
                      Into Subtotal = Sum(d.Quantity * d.UnitPrice *
                                         (1 - d.Discount)), LineItems = Count(),
                                         Freight = Sum(o.Freight) _
```

## Part II: Introducing Language Integrated Query

---

```
Select ShipCountry, LineItems, Subtotal, Freight  
  
' Substitute inline subquery in iterator to make freight deterministic  
For Each o In usOrdersTotaled6  
    Dim Ord = o ' Suppress complaint  
    Dim Freight = OrderList.Where(Function(n) n.ShipCountry =  
        Ord.ShipCountry).Sum(Function(f) f.Freight)  
    sbResult.Append(String.Format("Country = {0}, Orders = {1}, " + _  
        "LineItems = {2}, Subtotal = {3}, Freight = {4}, Total = {5}" + vbCrLf, _  
        o.ShipCountry, OrderList.Where(Function(n) n.ShipCountry =  
            Ord.ShipCountry).Count(), o.LineItems, o.Subtotal, Freight, _  
        o.Subtotal + Freight))
```

Next

*VB offers the option of specifying the source sequence for the Group By clause explicitly; you can substitute Group o By o.ShipCountry in the preceding expression.*

## **Combining Join and Group By Operations with Hierarchical Group Join Expressions**

The GroupBy() SQO doesn't join two collections, instead GroupBy() converts a single collection into a sequence of keys, such as ShipCountry in the preceding two examples, each paired with a collection of matching objects in that group. According to the February 2007 "Overview of Visual Basic 9.0" whitepaper, the "Group Join operator performs a grouped join of two collections based on matching keys extracted from the elements." The same is true for the GroupJoin() operator, but C# doesn't have a groupjoin keyword. When you add an into clause to a C# join statement the compiler substitutes GroupJoin() for Join(). The essence of GroupJoin() is that it delivers a hierarchical output of parent outer objects with a related group of inner child objects.

*Relational databases don't support hierarchical query outputs directly, so there's no equivalent T-SQL statement for the examples of the Group Join sections.*

The C# syntax for group joins is `join...seq2 in sourceSequence on seq1.key equals seq2key into seq3`. The important differences between the C# and VB syntaxes are:

- ❑ The C# LINQ query operator `group` doesn't prefix `join`.
- ❑ The C# `into` operator assigns the group sequence without the `= group` term.
- ❑ The C# outer sequence identifier must precede the inner sequence identifier in the `on...equals...` clause.

Following is a simple group join query against the OrderList and Order\_DetailsList collections, which returns only a single group:

## Chapter 4: Working with Advanced Query Operators and Expressions

---

### C# 3.0

```
var ukOrders7 = from o in OrderList
                 where o.ShipCountry == "UK"
                 join d in Order_DetailList
                   on o.OrderID equals d.OrderID into od
                 select new { order = o.OrderID, shipTo = o.ShipName,
                           lines = od.Count(), freight = o.Freight, details = od };

foreach (var o in ukOrders7)
{
    int items = 0;
    decimal subtotal = 0;
    foreach (var d in o.details)
    {
        items += d.Quantity;
        subtotal += d.Quantity * d.UnitPrice * (1m - (decimal)d.Discount);
    }
    sbResult.Append(String.Format("OrderID = {0}, ShipTo = {1}, LineItems = {2}, "+
        "Items = {3}, Subtotal = {4}, Freight = {5}, Total = {6}\r\n",
        o.order, o.shipTo, o.lines, items, subtotal, o.freight, subtotal +
        o.freight));
}
```

The VB syntax for a group join is `Group Join Seq1 In SourceSequence On Seq2Key Equals Seq1Key Into Seq3 Group`. The “handedness” of the Seq#Key values isn’t significant because both are in scope; the right-hand (outer) Seq2Key can be interchanged with the left-hand (inner) Seq1Key. `Group`, `Join`, and `Into` aren’t VB keywords, but `Seq3` must be assigned to `Group`.

In the following example, the `od.Count()` operator is called a *nested count*; the `Details` group is nested in the sequence of `Order` anonymous types:

### VB 9.0

```
Dim ukOrders7 = From o In OrderList _
                  Where o.ShipCountry = "UK" _
                  Group Join d In Order_DetailList _
                    On d.OrderID Equals o.OrderID Into od = Group _
                    Select Order = o.OrderID, ShipTo = o.ShipName, _
                           Lines = od.Count(), Freight = o.Freight, Details = od

For Each o In ukOrders7
    Dim Items As Integer = 0
    Dim Subtotal As Decimal = 0
    For Each d In o.Details
        Items += d.Quantity
        Subtotal += d.Quantity * d.UnitPrice * (1 - d.Discount)
    Next
    sbResult.Append(String.Format("OrderID = {0}, ShipTo = {1}, " & _
        "LineItems = {2}, "Items = {3}, Subtotal = {4}, Freight = {5}, " & _
        "Total = {6}" + vbCrLf, o.Order, o.ShipTo, o.Lines, Items, Subtotal, & _
        o.Freight, Subtotal + o.Freight))
Next
```

## Part II: Introducing Language Integrated Query

---

The `Details` group is a sequence so a nested iterator is required to format the string representation of `Details'` members.

Both queries return the following lines:

```
OrderID = 10289, ShipTo = B's Beverages, LineItems = 2, Items = 39,
Subtotal = 479.4000,
Freight = 22.7700, Total = 502.1700
OrderID = 10315, ShipTo = Island Trading, LineItems = 2, Items = 44,
Subtotal = 516.8000,
Freight = 41.7600, Total = 558.5600
OrderID = 10318, ShipTo = Island Trading, LineItems = 2, Items = 26,
Subtotal = 240.4000,
Freight = 4.7300, Total = 245.1300
OrderID = 10321, ShipTo = Island Trading, LineItems = 1, Items = 10,
Subtotal = 144.0000,
Freight = 3.4300, Total = 147.4300
```

## Comparing Group Joins with Nested LINQ Queries

Nested LINQ queries are an alternative to group joins but involve considerably more code to formulate an equivalent query. Nested queries incorporate queries to populate child sequences (groups) in their `Select()` projection list. Nested queries and group joins implicitly implement left outer joins; that is, a parent member is present when no corresponding child members exist.

Following are C# and VB versions of a three-level, hierarchical nested query and its nested iterator that returns lines from the `custs/Custs` sequence for UK customers and, if present, `orders/Orders` and `details/Details` groups:

### C# 3.0

```
var query = from c in CustomerList
            where c.Country == "UK"
            select new
            {
                custID = c.CustomerID,
                company = c.CompanyName,
                orders = from o in OrderList
                          where o.CustomerID == c.CustomerID
                          select new
                          {
                              orderID = o.OrderID,
                              orderDate = o.OrderDate,
                              shippedDate = o.ShippedDate,
                              freight = o.Freight,
                              details = from d in Order_DetailList
                                         where d.OrderID == o.OrderID
                                         select d
                          }
            }
```

## Chapter 4: Working with Advanced Query Operators and Expressions

---

```
};

foreach (var co in query)
{
    sbResult.Append(String.Format("CustomerID = {0}, Company = {1}\r\n",
        co.custID, co.company));

    if (co.orders.Count() > 0)
    {
        foreach (var o in co.orders)
        {
            sbResult.Append(String.Format("    OrderID = {0}, Order Date = {1},
                Shipped Date = {2}, Freight = {3}\r\n",
                o.orderID, o.orderDate, o.shippedDate, o.freight));
            foreach (var d in o.details)
            {
                decimal discDec = (decimal)d.Discount;
                sbResult.Append(String.Format("        SKU = {0}, Quan. = {1},
                    Price = {2}, Disc. = {3}, Ext. = {4}\r\n",
                    d.ProductID, d.Quantity, d.UnitPrice, d.Discount,
                    (d.Quantity * d.UnitPrice * (1m - discDec))));
            }
        }
    }
    else sbResult.Append("    No orders for " + co.custID + "\r\n");
}
```

Notice that there are no query syntax differences between the preceding C# and following VB versions of the nested query:

### VB 9.0

```
Dim Custs = From c In CustomerList _
    Where c.Country = "UK" _
    Select CustID = c.CustomerID, _
        Company = c.CompanyName, _
        Orders = From o In OrderList _
        Where o.CustomerID = c.CustomerID _
        Select OrderID = o.OrderID, _
            OrderDate = o.OrderDate, _
            ShippedDate = o.ShippedDate, _
            Freight = o.Freight, _
            Details = From d In Order_DetailList _
            Where d.OrderID = o.OrderID _
            Select d

For Each co In Custs
    sbResult.Append(String.Format("CustomerID = {0}, Company = {1}" & vbCrLf, _
        co.CustID, co.Company))
    If co.Orders.Count() > 0 Then
        For Each o In co.Orders
            sbResult.Append(String.Format("    OrderID = {0}, " & _
                "Order Date = {1}, Shipped Date = {2}, Freight = {3}" & vbCrLf, _
                o.OrderID, o.OrderDate, o.ShippedDate, o.Freight))
        
```

## Part II: Introducing Language Integrated Query

---

```
For Each d In o.Details
    sbResult.Append(String.Format("      SKU = {0}, Quan. = {1}, " & _
        "Price = {2}, Disc. = {3}, Ext. = {4}" & vbCrLf, _
        d.ProductID, d.Quantity, d.UnitPrice, d.Discount, _
        (d.Quantity * d.UnitPrice * (1 - d.Discount))))
Next
Next
Else
    sbResult.Append("    No orders for " & co.CustID & vbCrLf)
End If
Next
```

Following is the output of the two queries:

```
CustomerID = AROUT, Company = Around the Horn
    No orders for AROUT
CustomerID = BSBEV, Company = B's Beverages
    OrderID = 10289, Order Date = 8/26/2006 12:00:00 AM, Shipped Date = 8/28/1996
    12:00:00 AM, Freight = 22.77
        SKU = 3, Quan. = 30, Price = 8, Disc. = 0, Ext. = 240
        SKU = 64, Quan. = 9, Price = 26.6, Disc. = 0, Ext. = 239.4
CustomerID = CONSH, Company = Consolidated Holdings
    No orders for CONSH
...
CustomerID = SEVES, Company = Seven Seas Imports
    No orders for SEVES
```

## **Emulating Left Outer Joins with Entity Associations**

Sample code for emulating SQL-99 left outer joins with LINQ universally refers to the `GroupJoin()` S<sub>Q</sub>O. However, you don't need to use `join...into` or `Group Join...Into` expressions to return grouped results as sequences if your source entities have 1:many parent:child associations. For example, the following two expressions return a result identical to the two queries of the preceding section:

### **VB 9.0**

```
var custs2 = from co in CustomerList
    where co.Country == "UK"
    select new { custID = co.CustomerID, company = co.CompanyName,
        orders = co.Orders };
```

### **VB 9.0**

```
Dim Custs2 = From co In CustomerList _
    Where co.Country = "UK" _
    Select CustID = co.CustomerID, Company = co.CompanyName, _
        Orders = co.Orders
```

## Chapter 4: Working with Advanced Query Operators and Expressions

---

The iterator does most of the work when processing queries that emulate left out joins. The iterator code isn't shown because it's the same as that for the preceding two samples, except:

- ❑ **C#:** Change `o.orderID, o.orderDate, o.shippedDate, o.freight` to `o.OrderID, o.OrderDate, o.ShippedDate, o.Freight`, and change `o.details` to `o.Order_Details` to reflect the change from named projections to members.
- ❑ **VB:** Change `o.details` to `o.Order_Details`.

Replacing group joins with associations provides a substantial performance improvement if the sequences you're joining contain many members.

*Always take advantage of associations between related entities. If your objects don't have associations, perhaps as the result of serialization for transport across process boundaries, you might find it more efficient to add the associations before performing extensive join operations on large business entities.*

## Taking Full Advantage of the Contains() SQO

Chapter 3's *Contains* section covers the basic definition of the `Contains()` Standard Query Operator. The `Contains()` operator returns `True` if a sequence contains a specified element. However, the `Contains()` operator also can be used to return `True` if a value matches one or more elements of a sequence. The following sections show examples of SQL `WHERE` clauses and `IN()` operators that you can emulate with the `Contains()` operator.

### Emulating SQL Where Clauses with Compound OR Operators

LINQ query expressions and method call syntax have no problem handling the LINQ equivalent of SQL `WHERE` clauses with `AND` operators which narrow the filter restriction, such as:

#### T-SQL

```
SELECT * FROM Orders WHERE ShipCountry = 'USA' AND ShipDate >= '1/1/2008'
```

with these query expressions:

#### C# 3.0

```
var orders = from o in orderList where o.ShipCountry == "USA" &&
    o.ShippedDate >= DateTime.Parse("1/1/2008") select o;
```

#### VB 9.0

```
Dim usOrders = From o In OrderList Where o.ShipCountry = "USA" And _
    o.ShippedDate >= "1/1/2008" Select o
```

## Part II: Introducing Language Integrated Query

---

and corresponding method calls:

### C# 3.0

```
var orders = orderList.Where(o => o.ShipCountry == "USA").  
Where(o => o.ShippedDate >= DateTime.Parse("1/1/2008"));
```

### VB 9.0

```
Dim usOrders = OrderList.Where(Function(o) o.ShipCountry = "USA") .  
Where(Function(o) o.ShippedDate >= "1/1/2008")
```

However, neither of the following LINQ to Objects expressions will compile because they can't be composed into method calls that broaden the filter restriction:

### C# 3.0

```
var pacificCusts = from c In customerList where c.Region == "CA" ||  
c.Region == "OR" || c.Region == "WA" select c;
```

### VB 9.0

```
Dim PacificCusts = From c In CustomerList Where c.Region = "CA" Or  
c.Region = "OR" Or c.Region = "WA" Select c
```

What's missing from LINQ to Objects is an `OrWhere` SQO. The preceding two expressions are valid in LINQ to SQL queries because an expression tree handles the translation to T-SQL.

The `Contains()` source usually is an `IEnumerable<TSource>` sequence returned by a LINQ query and the operator's argument ordinarily takes a single object or literal value of the same type (`TSource`). However, you can substitute an `IEnumerable<TSource>` sequence that you create from an array of literal values with instructions and a method call such as:

### C# 3.0

```
string[] strStates = { "CA", "OR", "WA" }();  
var states = from st in strStates select st;  
Dim pacificCusts = customerList.Where(c => states.Contains(c.Region));
```

### VB 9.0

```
Dim strStates As String() = {"CA", "OR", "WA"}  
Dim States = From St In strStates Select St  
Dim PacificCusts = CustomerList.Where(Function(c) States.Contains(c.Region))
```

An alternative to using `Contains()` is the `Union()` SQO, but query processing is considerably less efficient:

### C# 3.0

```
var custsCaOr = CustomerList.Where(c => c.Region == "CA").  
Union(CustomerList.Where(c => c.Region == "OR"));  
var pacificCusts2 = custsCaOr.Union(CustomerList.Where(c => c.Region == "WA"));
```

### VB 9.0

```
Dim CustsCaOr = CustomerList.Where(Function(c) c.Region = "CA"). _
    Union(CustomerList.Where(Function(c) c.Region = "OR"))
Dim CustsCaOrWa = CustsCaOr.Union(CustomerList.Where(Function(c) c.Region = "WA"))
```

## ***Emulating the SQL IN() Function with Contains()***

The SQL `IN()` function returns TRUE if the value of the expression it tests is contained within a single-column resultset or a comma-separated set of literal values. The data type of the expression and the column or literal values must be the same. `Contains<T>()` accepts a `Sequence<T>` as its argument and returns `true` if any member matches the test expression's value; `Contains<T>()` doesn't support literals or array initializers as its argument.

There's no inherent difference between emulating the SQL `IN()` function or Or'd `Where` clauses, but it's worthwhile demonstrating that the `Contains<T>()` accepts a `Sequence<T>` that can be a value type or an `Enumerable` projection from a complex reference type. For example, the following expressions return the same list of five Northwind employees with odd `EmployeeID`s:

### C# 3.0

```
int[] emplIDs = {1, 3, 5, 7, 9};
var empls = from em in emplIDs select em;
var oddEmpls = EmployeeList.Where(m => empls.Contains(m.EmployeeID));

foreach (var m in oddEmpls)
{
    sbResult.Append(String.Format("ID = {0}, LastName = {1}, FirstName = {2},
        Country = {3}\r\n",
        m.EmployeeID, m.LastName, m.FirstName, m.Country));
}
```

### VB 9.0

```
Dim intEmplIDs As Integer() = {1, 3, 5, 7, 9}
Dim Emps = From Em In intEmplIDs Select Em
Dim OddEmpls = EmployeeList.Where(Function(m) Emps.Contains(m.EmployeeID))

For Each m In OddEmpls
    sbResult.Append(String.Format("ID = {0}, LastName = {1}, " & _
        "FirstName = {2}, Country = {3}" + vbCrLf, _
        m.EmployeeID, m.LastName, m.FirstName, m.Country))
Next
```

## Part II: Introducing Language Integrated Query

---

These method calls and expressions return the same list of Northwind products in categories that start with the letter "C":

### C# 3.0

```
var cats = CategoryList.Where(c => c.CategoryName.StartsWith("C")).  
    Select(c => c.CategoryID);  
var prodsInC = ProductList.Where(c => cats.Contains((int)c.CategoryID)).  
    OrderBy(p => p.CategoryID).ThenBy(p => p.ProductID);  
  
foreach(var p in prodsInC)  
{  
    sbResult.Append(String.Format("CatID = {0}, CatName = {1}, ProdID = {2},  
        ProdName = {3}, Stock = {4}\r\n", p.CategoryID, p.Category.CategoryName,  
        p.ProductID, p.ProductName, p.UnitsInStock));  
}
```

### VB 9.0

```
Dim Cs = From c In CategoryList Where c.CategoryName Like "C*" Select c.CategoryID  
Dim ProdsInC = ProductList.Where(Function(c) Cs.Contains(c.CategoryID)). _  
    OrderBy(Function(p) p.CategoryID).ThenBy(Function(p) p.ProductID)  
  
For Each p In ProdsInC  
    sbResult.Append(String.Format("CatID = {0}, CatName = {1}, ProdID = {2}, " & _  
        "ProdName = {3}, Stock = {4}" + vbCrLf, p.CategoryID, _  
        p.Category.CategoryName, p.ProductID, p.ProductName, p.UnitsInStock))  
Next
```

The `Like` expression keyword is specific to VB 9.0 and isn't available in C# 3.0. VB 9.0 enables comparison of nullable (`p.CategoryID`) with non-nullables (`c.CategoryID`) types, even with Option Strict On.

## Compiling Query Expression Trees to Improve Performance

LINQ implementations that translate LINQ queries to another query language use expression trees to construct a command string in the query language. Examples of LINQ implementations that generate domain-specific query strings are LINQ to SQL for T-SQL statements, LINQ to Entities for Entity SQL queries, LINQ to SharePoint for Collaborative Application Markup Language (CAML), and LINQ to Astoria for URI-based queries. Chapters later in the book cover all the preceding implementations.

The compiled code that translates a LINQ query to a domain-specific query language, such as T-SQL or Entity SQL, is commonly called the *query pipeline*. The LINQ to SQL query pipeline, which includes a very large expression tree, involves about 15 individual steps. Building the query pipeline for even

## Chapter 4: Working with Advanced Query Operators and Expressions

---

a simple LINQ query and performing the translation is very resource intensive. The process significantly degrades performance compared to a T-SQL query that returns a `DataReader` directly. The following LINQ to SQL query against the Northwind Orders table executes 1,000 times in about 6.45 seconds with USA as the `country` parameter value:

```
nwdc.Orders.Where(o => o.ShipCountry == country).OrderByDescending(o => o.OrderID)
```

Parameterized queries against databases, such as SQL Server 200x improve performance by caching the query plan for repeated execution with changes only to parameter values. LINQ to SQL and LINQ to Entities let you compile parameterized LINQ queries for better performance. As an example, the compiled version of the preceding query executes 1,000 times in about 1.45 seconds, a 445 percent performance boost.

*SQL Server 6.5 and earlier versions compiled parameterized queries into stored procedures to improve performance, but there's no difference between SQL Server 7.0 and later's performance of parameterized stored procedures and queries.*

The process for compiling domain-specific queries is particular to the domain, but LINQ to SQL and LINQ to Entities share a common technique and syntax. You declare a `static/Shared CompiledQuery.Compile()` function that returns a delegate to call with the appropriate set of arguments. Here are the four LINQ to SQL `CompiledQuery.Compile()` overload signatures for zero, one, two, and three parameters:

### C# 3.0

```
public static Func<TArg0, TResult> Compile<TArg0, TResult>(
    Expression<Func<TArg0, TResult>> query
) where TArg0 : DataContext

public static Func<TArg0, TArg1, TResult> Compile<TArg0, TArg1, TResult>(
    Expression<Func<TArg0, TArg1, TResult>> query
) where TArg0 : DataContext

public static Func<TArg0, TArg1, TArg2, TResult>
    Compile<TArg0, TArg1, TArg2, TResult>(
        Expression<Func<TArg0, TArg1, TArg2, TResult>> query
) where TArg0 : DataContext

public static Func<TArg0, TArg1, TArg2, TArg3, TResult>
    Compile<TArg0, TArg1, TArg2, TArg3, TResult>(
        Expression<Func<TArg0, TArg1, TArg2, TArg3, TResult>> query
) where TArg0 : DataContext
```

`TArg0` is either a LINQ to SQL `DataContext` or LINQ to Entities `ObjectContext` type.

## Part II: Introducing Language Integrated Query

---

### VB 9.0

```
Public Shared Function Compile(Of TArg0 As DataContext, TResult) ( _
    Query As Expression(Of Func(Of TArg0, TResult)) _ 
) As Func(Of TArg0, TResult)

Public Shared Function Compile(Of TArg0 As DataContext, TArg1, TResult) ( _
    Query As Expression(Of Func(Of TArg0, TArg1, TResult)) _ 
) As Func(Of TArg0, TArg1, TResult)

Public Shared Function Compile(Of TArg0 As DataContext, TArg1, TArg2, TResult) ( _
    Query As Expression(Of Func(Of TArg0, TArg1, TArg2, TResult)) _ 
) As Func(Of TArg0, TArg1, TArg2, TResult)

Public Shared Function Compile(Of TArg0 As DataContext, TArg1, TArg2, TArg3, _
    TResult) ( _
    Query As Expression(Of Func(Of TArg0, TArg1, TArg2, TArg3, TResult)) _ 
) As Func(Of TArg0, TArg1, TArg2, TArg3, TResult)
```

The signatures for compiling LINQ to Entities queries are the same as the preceding, except for replacement of `DataContext` with `ObjectContext` as the type of `TArg0`. It's likely that third-party LINQ implementations offering a compilation option will adopt similar syntax.

*The sample projects for this section are `LINQtoSQLPerfCS.sln` and `LINQtoSQLPerfVB.sln` in the `\WROX\ADONET\Chapter4\CS` and ... \VB folders, respectively. These WinForm test harnesses expect the Northwind sample database to be attached to a local instance of SQL Server 2005+ Express. If your configuration differs, fix the connection string in the app.config file.*

Following are examples of the delegate functions for the LINQ query at the beginning of this section:

### C# 3.0

```
static class Queries
{
    public static Func<NwindOrdersDataContext, string, IOrderedQueryable<Order>>
        OrdersByCountry = CompiledQuery.Compile((NwindOrdersDataContext nwdc,
            string country)
        => nwdc.Orders.Where(o => o.ShipCountry == country) .
            OrderByDescending(o => o.OrderID));
}
```

### VB 9.0

```
Public Shared OrdersByCountry As Func(Of NwindOrdersDataContext, String, _
    IOrderedQueryable(Of Order)) = _
    CompiledQuery.Compile(Function(dcNwind As NwindOrdersDataContext, _
        Country As String) _
        dcNwind.Orders.Where(Function(o) o.ShipCountry = Country). _
        OrderByDescending(Function(o) o.OrderID))
```

## Chapter 4: Working with Advanced Query Operators and Expressions

Here are methods that invoke the delegates:

### C# 3.0

```
private IEnumerable<Order> GetOrdersByCountry(string country)
{
    return Queries.OrdersByCountry(dcOrders, country);
}
```

### VB 9.0

```
Private Function GetOrdersByCountry(ByVal Country As String) _
As IEnumerable(Of Order)
    Return Queries.OrdersByCountry(dcOrders, Country)
End Function
```

And here are the calls to the GetOrdersByCountry method:

### C# 3.0

```
IEnumerable<OrderAll> compiledQuery =
    GetOrdersByCountry(country).AsEnumerable<Order>();
```

### VB 9.0

```
Dim CompiledQuery As IEnumerable(Of Order) = _
    GetOrdersByCountry(strCountry).AsEnumerable()
```

*You can't store queries that return an anonymous type in static variables because an anonymous type has no name to use as the <t>/Of T argument value.*

The two test harnesses measure the time to execute 500 instances of a LINQ to SQL or T-SQL query with a projection or SELECT list that returns OrderID, CustomerID, EmployeeID, ShippedDate, and ShipCountry property values, which total an average 36 bytes. Figure 4-2 shows the C# test harness, which has a few more features than the VB implementation.

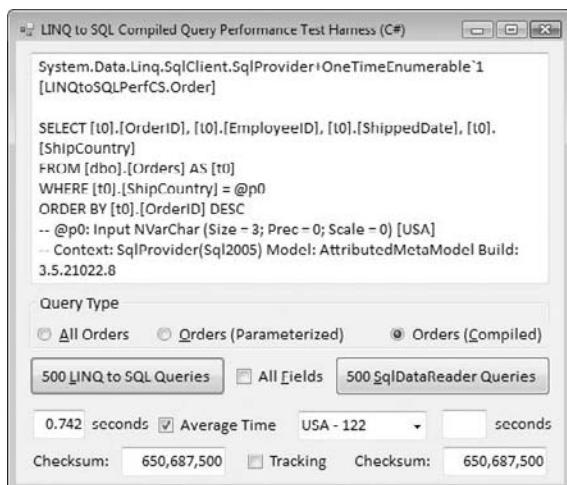


Figure 4-2

## Part II: Introducing Language Integrated Query

---

The `System.Data.Linq.SqlClient.SqlProvider+OneTimeEnumerable`1[LINQtoSQLPerfCS.Order]` at the top of the text box is the typical output of the `query.AsQueryable().Expression.ToString()` method for a compiled query. `OneTimeEnumerable` indicates that you can only iterate a compiled query once, which is the reason for re-creating it prior to iterating its result in the preceding `GetOrdersByCountry()` method calls.

The following table presents times (in seconds) for 500 iterations of a nonparameterized LINQ and T-SQL `SqlDataReader` query for all 830 Northwind orders, as well as parameterized and compiled LINQ queries and T-SQL queries for varying number of instances or rows and the ratio of execution times for compiled LINQ and T-SQL `SqlDataReader` queries. Order instances have `OrderID`, `CustomerID`, `EmployeeID`, `ShippedDate`, and `ShipCountry` properties only, which results in an average 36-byte payload with Unicode `CustomerID` and `ShipCountry` encoding. The table demonstrates that the benefit of LINQ query compilation increases as the number of Order instances increases; at 122 instances the performance penalty for a LINQ query is only about 21% of the `SqlDataReader` execution time.

Orders:	All 830	122 US	56 UK	28 Italy	13 Port.	6 Norway
No Param	2.022					
Parameterized		2.637	2.552	2.508	2.471	2.455
Compiled		0.725	0.667	0.619	0.601	0.608
SqlDataReader	2.056	0.600	0.450	0.416	0.376	0.363
Compiled / SqlDataReader		1.208	1.482	1.488	1.598	1.675

*Notice the substantial performance penalty (more than 30 percent) for adding a `Where` filter to the query.*

*Data in the preceding table was obtained under Visual Studio 2008 in a Windows Vista virtual machine with 1GB RAM running under Windows Server 2003 R2 on a Gateway S5200D machine with a dual-core Pentium 2.8 MHz processor with a 160-GB SATA disk.*

As a general rule, it's advantageous to compile static or parameterized LINQ queries for production applications of implementations that offer the `CompiledQuery.Compile()` or equivalent feature.

## Mocking Collections for Testing LINQ to SQL and LINQ to Entities Projects

The traditional approach to software development is to define the requirements for a program and its features, write the source code to implement the features, and then perform tests of various use cases to determine whether the implementation meets current requirements. Test-driven development (TDD) is a recent approach to creating applications by writing multiple, automated *unit tests* for a feature's use cases before writing code to implement the feature. After designing the unit test suite to satisfy expected

## Chapter 4: Working with Advanced Query Operators and Expressions

---

use cases, developers write code to pass the tests. An important feature of TDD is that all tests must fail before the developer writes any code to implement the feature. Developers run the unit tests each time they make changes to the code. Tests are added or altered to conform to changing requirements and special cases discovered during testing. After the feature passes all tests, it's a common practice to *refactor* its source code to remove duplication, conform to formatting and naming conventions, and remove unnecessary development or testing artifacts. VS 2008 Professional Edition includes most of the unit testing features of VS 2008 Team System. NUnit ([www.nunit.org](http://www.nunit.org)) and MBUnit ([www.mbunit.com](http://www.mbunit.com)) are popular open-source unit test frameworks for .NET.

TDD assumes use of a source-code version control system, such as VS 2008 Team Foundation Server, SourceGear Vault, or Subversion. Repeated tests are made during the refactoring process; when a test fails, changes to the source code are rolled back to the point where the code again passes all tests. Then the previous changes are examined to determine what caused the failure. The code is then refactored and retested. TDD is an iterative process, so the tests ordinarily execute every time the developer builds and runs the project. TDD is a primary principle of Extreme Programming (XP) and agile software development, both of which rely on iterative development methodologies.

Most applications with the scope and complexity that warrant automated unit testing involve business objects instantiated from and persisted (saved) to database tables by an object/relational mapping (O/RM) tool. LINQ to SQL, the subject of the next chapter, and the Entity Framework, the topic of this book's second half, are O/RM tools that enable LINQ queries. Retrieving a representative number of records from many related tables to generate a complete business object graph for unit tests can consume an extraordinary amount of resources and slow the unit testing process to a crawl. In addition, object persistence and entity manipulation are distinctly different activities; testing the two activities in combination violates the principle of separation of concerns with unit tests.

Creating in-memory collections of *mock objects* from class definitions and object initializers stored in the local file system instead of a relational database can increase developers' productivity dramatically. *Mock objects* mimic objects supplied to the project under test by an O/RM tool and eliminate side effects introduced by the O/RM tool from the test regimen. Using mock objects eliminates the need to initialize altered database values, such as time stamps, between tests. Rhino Mocks ([www.codeproject.com/useritems/Rhino\\_Mocks.asp](http://www.codeproject.com/useritems/Rhino_Mocks.asp)) and NMock2 (<http://sourceforge.net/projects/nmock>) are open-source, and TypeMocks ([www.typemock.com](http://www.typemock.com)) is a commercial mock object framework.

*Martin Fowler's "Mocks Aren't Stubs" paper ([www.martinfowler.com/articles/mocksArentStubs.html](http://www.martinfowler.com/articles/mocksArentStubs.html)) offers a concise explanation of how mock objects work.*

### **Creating Mock Object Classes and Initializers**

Chapter 3's "Sample Classes for LINQ to Objects Code Examples" section briefly describes simple class files and collection initializers generated by an early version of the LINQ In-Memory Object Generation Utility (LIMOG.sln). The later version of LIMOG that's in the \WROX\ADONET\Chapter\LIMOGUtility folder is a utility that generates code for creating entity classes, optional parameterless and parameterized constructors, and object intializers for creating generic collections. LIMOG's current implementation generates C# or VB code for these objects from any database installed on a local instance of SQL Server 2005+ Express. As usual, you can alter the connection string in the app.config file to suit alternative SQL Server versions or locations.

## Part II: Introducing Language Integrated Query

LIMOG has been tested extensively only with the Northwind, AdventureWorks, AdventureWorksLT, and OakLeafU databases. You might need to modify entity class code after generation if you use other databases as a data source.

Figure 4-3 shows LIMOG's UI in the first stage of generating code for an Order entity from Northwind's Orders table. The input box lets you accept or specify the primary key constraint value (OrderID) with which to search other tables for relationships. In this case, the search finds OrderID as member of the Order Details table's composite primary key for a 1:many relationship (EntitySet).

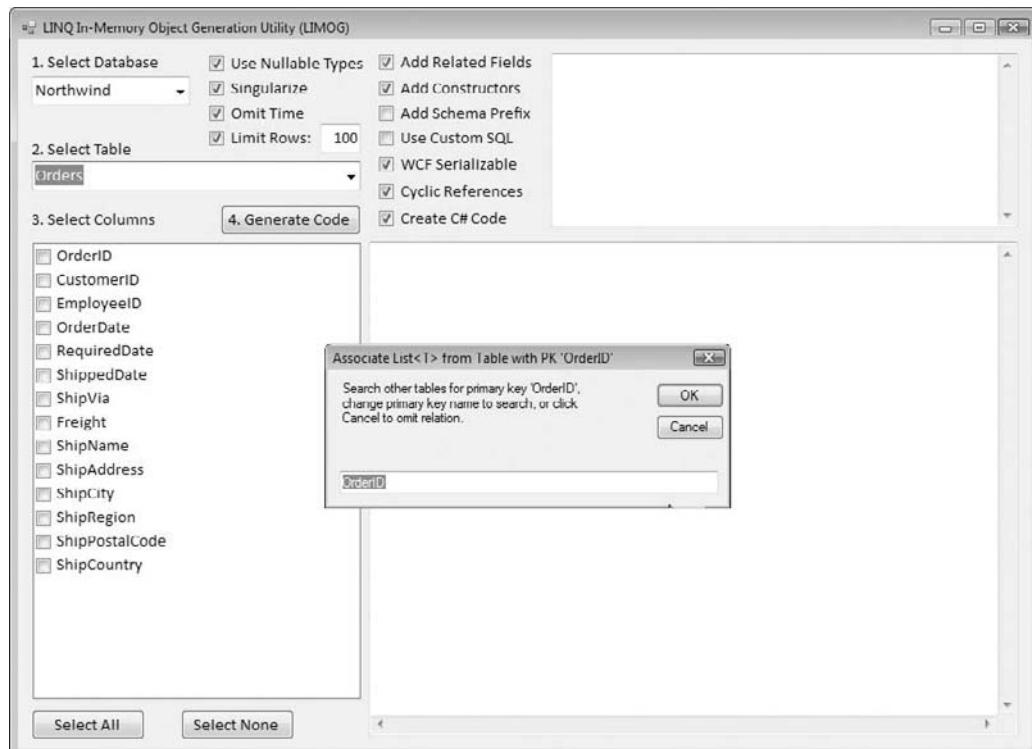


Figure 4-3

A similar search looks for primary key names that correspond to the table's foreign-key names, CustomerID, EmployeeID, and ShipperID for this example. Figure 4-4 shows LIMOG's UI after completing the code generation process, which has added the Order\_Details, Customer, Employee, and Shipper members to the bottom of the Columns list. You then copy and paste the code from the text boxes to your C# or VB classes.

## Chapter 4: Working with Advanced Query Operators and Expressions

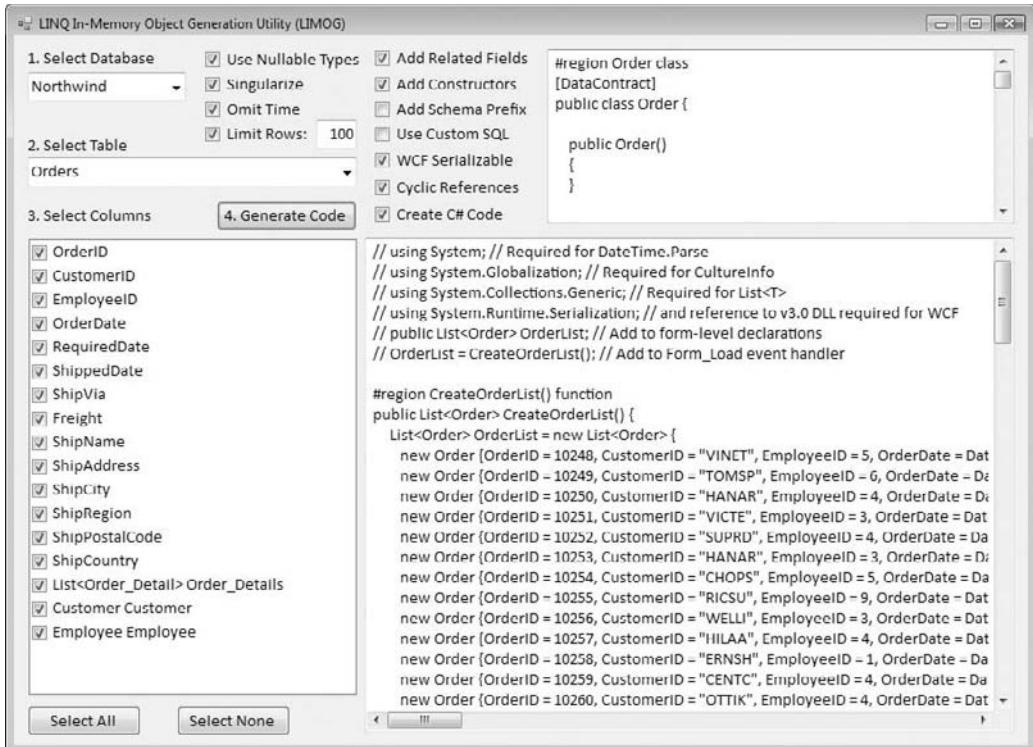


Figure 4-4

Class decoration for binary serialization with `[Serializable]/<Serializable()>` attributes is the default because binary serialization preserves associations. However, you can substitute .NET 3.5's `DataContractSerializer` (DCS) for industry standard (WS-\*<sup>1</sup>) SOAP serialization with Windows Communication Foundation (WCF) by marking the `WCF Serializable` check box. DCS doesn't support serializing classes with cyclic relationships generated by 1:many (`Order_Details` in Figure 4-4) and many:1 relationships (`Customer`, `Employee`, `Shipper`). Such classes require the `NetDataContractSerializer` (NDCS) to create nonstandard XML messages that use `Id` and `Ref` attributes to emulate object references and prevent associations from generating infinite loops.

Databinding entities to components and controls requires public properties with getters and setters; C# 3.0's new automatic properties makes property code much less verbose than VB 9.0's traditional

## Part II: Introducing Language Integrated Query

---

property declarations. Following are examples of a C# 3.0 class that specifies binary serialization and a VB 9.0 class that works with WCF's DCS or NDCS:

### C# 3.0

```
[Serializable]
public class Order_Detail {

    public Order_Detail()
    {
    }

    public Order_Detail(int orderID,
                        int productID,
                        decimal unitPrice,
                        short quantity,
                        float discount,
                        Order order,
                        Product product)
    {
        this.OrderID = orderID;
        this.ProductID = productID;
        this.UnitPrice = unitPrice;
        this.Quantity = quantity;
        this.Discount = discount;
        this.Order = order;
        this.Product = product;
    }

    public int OrderID { get; set; }
    public int ProductID { get; set; }
    public decimal UnitPrice { get; set; }
    public short Quantity { get; set; }
    public float Discount { get; set; }
    public Order Order { get; set; }
    public Product Product { get; set; }
}
```

### VB 9.0

```
<DataContract(Name := "Order_Detail", Namespace := "")> _
Public Class Order_Detail

    Public Sub New()
    End Sub

    Public Sub New(ByVal intOrderID As Integer, _
                  ByVal intProductID As Integer, _
                  ByVal decUnitPrice As Decimal, _
                  ByVal shQuantity As Short, _
                  ByVal sngDiscount As Single, _
                  ByVal objOrder As Order, _
                  ByVal objProduct As Product)
        Me.OrderID = intOrderID
        Me.ProductID = intProductID
    End Sub
}
```

## Chapter 4: Working with Advanced Query Operators and Expressions

---

```
    Me.UnitPrice = decUnitPrice
    Me.Quantity = shtQuantity
    Me.Discount = sngDiscount
    Me.Order = objOrder
    Me.Product = objProduct
End Sub

Private _OrderID As Integer
<DataMember(Name := "OrderID", Order:= 1)> _
Public Property OrderID() As Integer
    Get
        Return Me._OrderID
    End Get
    Set(ByVal value As Integer)
        Me._OrderID = value
    End Set
End Property

Private _ProductID As Integer
<DataMember(Name := "ProductID", Order:= 2)> _
Public Property ProductID() As Integer
    Get
        Return Me._ProductID
    End Get
    Set(ByVal value As Integer)
        Me._ProductID = value
    End Set
End Property

Private _UnitPrice As Decimal
<DataMember(Name := "UnitPrice", Order:= 3)> _
Public Property UnitPrice() As Decimal
    Get
        Return Me._UnitPrice
    End Get
    Set(ByVal value As Decimal)
        Me._UnitPrice = value
    End Set
End Property

Private _Quantity As Short
<DataMember(Name := "Quantity", Order:= 4)> _
Public Property Quantity() As Short
    Get
        Return Me._Quantity
    End Get
    Set(ByVal value As Short)
        Me._Quantity = value
    End Set
End Property

Private _Discount As Single
<DataMember(Name := "Discount", Order:= 5)> _
```

## Part II: Introducing Language Integrated Query

---

```
Public Property Discount() As Single
    Get
        Return Me._Discount
    End Get
    Set(ByVal value As Single)
        Me._Discount = value
    End Set
End Property

Private _Order As Order
<DataMember(Name := "Order", Order:= 6)> _
Public Property Order() As Order
    Get
        Return Me._Order
    End Get
    Set(ByVal value As Order)
        Me._Order = value
    End Set
End Property

Private _Product As Product
<DataMember(Name := "Product", Order:= 7)> _
Public Property Product() As Product
    Get
        Return Me._Product
    End Get
    Set(ByVal value As Product)
        Me._Product = value
    End Set
End Property
End Class
```

The VB version shows the `<DataContract>` attribute for the class and `<DataMember>` attributes applied to members. The parameters are optional, although `Order = n` or `Order:= n` is required to preserve member sequence during serialization/deserialization operations.

*You'll also find creating or re-creating object associations to be quite important when serializing and deserializing object graphs for cross-domain transport in service-oriented projects.*

## **Creating Object Graphs with GroupJoin Expressions**

The unit-test code for the project executes sets of C# functions or VB procedures and functions to instantiate and populate the property values of mock classes. However, populating associations requires the test class to execute custom code. Objects are reference types; this means there is only one instance of a particular entity in memory at one time. Adding objects to many:1 or 1:many associations creates references to objects that are already in memory; the use of references instead of multiple instances of the same object is what prevents circular references from creating infinite loops of associations.

The NorthwindObjectsCS and NorthwindObjectsVB projects (in `\WROX\ADONET\Chapter04` subfolders of the same name) demonstrate how to create entity collections from LIMOG-generated class and list code. They also let you test the effect of binary serialization and deserialization on associations.

## Chapter 4: Working with Advanced Query Operators and Expressions

Figure 4-5 shows the projects' UI after clicking the Load (Initializer) to generate seven entity collections and the Orders Associations button. The latter button adds a 1:many association between the Order and Order\_Detail entities and many:1 associations between Order and Customer, Employee, and Shipper entities.

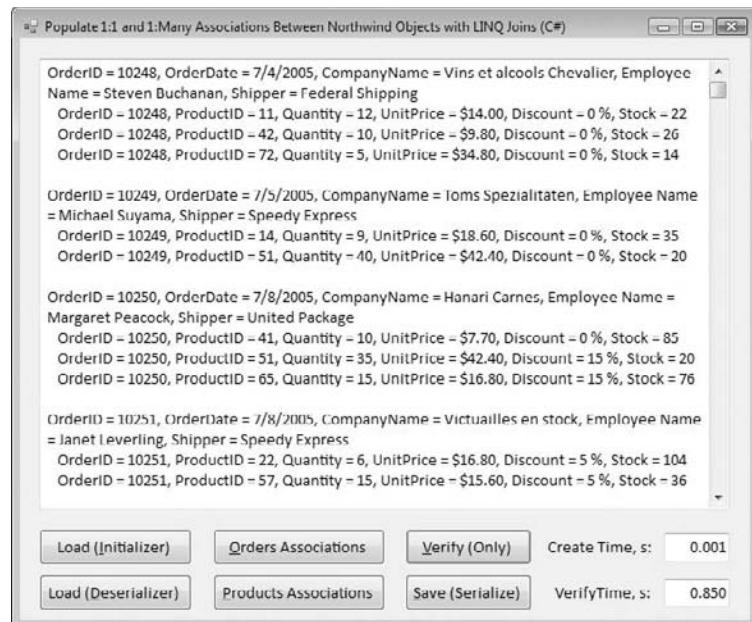


Figure 4-5

Following are the LINQ group join/Group Join expressions that enable adding the Orders associations with the fewest lines of code:

### C# 3.0

```
private void btnOrders_Click(object sender, EventArgs e)
{
    // Generate Order associations from joined Customer, Employee, Shipper,
    // and Order_Detail instances
    var query1 = from o in OrderList
                join c in CustomerList on o.CustomerID equals c.CustomerID into co
                join m in EmployeeList on o.EmployeeID equals m.EmployeeID into eo
                join s in ShipperList on o.ShipVia equals s.ShipperID into so
                join d in Order_DetailList on o.OrderID equals d.OrderID into od
                select new { Order = o, Customer = co, Employee = eo, Shipper = so,
                            Order_Detail = od };

    foreach (var o in query1)
```

## Part II: Introducing Language Integrated Query

---

```
{  
    // Add 1:1 associations  
    o.Order.Customer = o.Customer.ElementAt(0);  
    o.Order.Employee = o.Employee.ElementAt(0);  
    o.Order.Shipper = o.Shipper.ElementAt(0);  
  
    // Clear Order_Details before adding associations  
    o.Order.Order_Details.Clear();  
  
    // Populate 1:many association  
    foreach (var d in o.Order_Detail)  
        o.Order.Order_Details.Add(d);  
    }  
  
    // Populate Orders for each Customer  
    var query2 = from c in CustomerList  
        join o in OrderList on c.CustomerID equals o.CustomerID into oc  
        select new { Customer = c, CustOrders = oc };  
  
    foreach (var c in query2)  
    {  
        // Clear Orders list before adding associations  
        c.Customer.Orders.Clear();  
  
        // Populate 1:many association  
        foreach (var oc in c.CustOrders)  
            c.Customer.Orders.Add(oc);  
    };  
  
    // Populate Orders for each Employee  
    var query3 = from m in EmployeeList  
        join o in OrderList on m.EmployeeID equals o.EmployeeID into oe  
        select new { Employee = m, EmplOrders = oe };  
  
    foreach (var m in query3)  
    {  
        // Clear Orders list before adding associations  
        m.Employee.Orders.Clear();  
  
        // Populate 1:many association  
        foreach (var oe in m.EmplOrders)  
            m.Employee.Orders.Add(oe);  
    };  
  
    // Populate Orders for each Shipper  
    var query4 = from s in ShipperList  
        join o in OrderList on s.ShipperID equals o.ShipVia into os  
        select new { Shipper = s, ShipOrders = os };  
  
    foreach (var s in query4)
```

## Chapter 4: Working with Advanced Query Operators and Expressions

---

```
{  
    // Clear Orders list before adding associations  
    s.Shipper.Orders.Clear();  
  
    // Populate 1:many association  
    foreach (var os in s.ShipOrders)  
        s.Shipper.Orders.Add(os);  
};  
}
```

### VB 9.0

```
Private Sub btnOrders_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)  
Handles btnOrders.Click  
    Dim query1 = From o In OrderList _  
        Group Join c In CustomerList On o.CustomerID Equals _  
            c.CustomerID Into co = Group _  
        Group Join m In EmployeeList On o.EmployeeID Equals _  
            m.EmployeeID Into eo = Group _  
        Group Join s In ShipperList On o.ShipVia Equals _  
            s.ShipperID Into so = Group _  
        Group Join d In Order_DetailList On o.OrderID Equals _  
            d.OrderID Into od = Group _  
        Select New With {.Order = o, .Customer = co, .Employee = eo, _  
            .Shipper = so, .Order_Detail = od}  
  
    For Each o In query1  
        ' Add many:1 associations  
        o.Order.Customer = o.Customer.ElementAt(0)  
        o.Order.Employee = o.Employee.ElementAt(0)  
        o.Order.Shipper = o.Shipper.ElementAt(0)  
  
        ' Clear Order_Details before adding associations  
        o.Order.Order_Details.Clear()  
  
        ' Populate 1:many association  
        For Each d In o.Order_Detail  
            o.Order.Order_Details.Add(d)  
        Next d  
    Next o  
  
    ' Populate Orders for each Customer  
    Dim query2 = From c In CustomerList _  
        Group Join o In OrderList On c.CustomerID Equals _  
            o.CustomerID Into oc = Group _  
        Select New With {.Customer = c, .CustOrders = oc}  
  
    For Each c In query2  
        ' Clear Orders list before adding associations  
        c.Customer.Orders.Clear()  
  
        ' Populate 1:many association
```

## Part II: Introducing Language Integrated Query

---

```
For Each oc In c.CustOrders
    c.Customer.Orders.Add(oc)
Next oc
Next c

' Populate Orders for each Employee
Dim query3 = From m In EmployeeList _
    Group Join o In OrderList On m.EmployeeID Equals _
        o.EmployeeID Into oe = Group _
    Select New With {.Employee = m, .EmplOrders = oe}

For Each m In query3
    ' Clear Orders list before adding associations
    m.Employee.Orders.Clear()

    ' Populate 1:many association
    For Each oe In m.EmplOrders
        m.Employee.Orders.Add(oe)
    Next oe
Next m

' Populate Orders for each Shipper
Dim query4 = From s In ShipperList _
    Group Join o In OrderList On s.ShipperID Equals _
        o.ShipVia Into os = Group _
    Select New With {.Shipper = s, .ShipOrders = os}

For Each s In query4
    ' Clear Orders list before adding associations
    s.Shipper.Orders.Clear()

    ' Populate 1:many association
    For Each os In s.ShipOrders
        s.Shipper.Orders.Add(os)
    Next os
Next s
End Sub
```

Clearing existing orders before adding associations is required when hydrating objects with binary or DCS deserialization. These two deserializers produce multiple instances (*deep clones*) of identical objects rather than *shallow clones*, which don't include fields or properties for associated entities. In fact, binary serialization and deserialization, usually to and from a `MemoryStream` object, is the only practical method to create a deep clone in memory. Duplication of objects having identical property values (indicated by `object1.Equals(object2) = true`) consumes additional memory and creates ambiguous object versions. You can determine if two objects point to the same instance by testing if `object1.ReferenceEquals(object2) = true`.

The `ObjectGraphCS` and `ObjectGraphVB` sample projects in the same-named subfolders demonstrate that LIMOG-generated collections can serve as the data source for databound components and controls and that code such as the preceding substitutes references for duplicate instances. Figure 4-6 shows the `ObjectGraph` projects' UI, which includes `DataGridView` controls populated by `Customer.Orders` and `Order.Order_Details` property values.

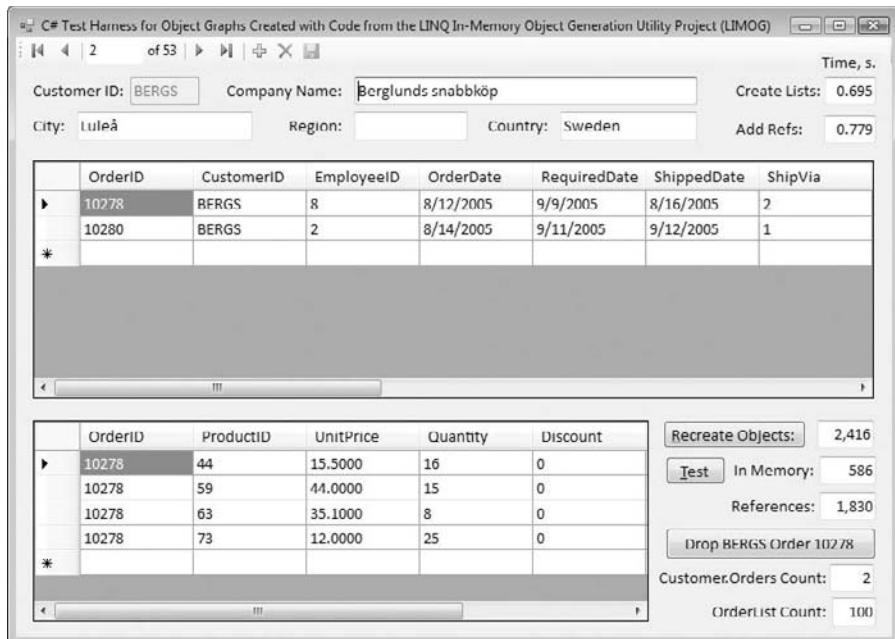


Figure 4-6

Of the 2,416 objects and associations created by code similar to that of the NwindObjects, 586 are in memory and more than three times as many, 1,830, are represented by references.

## Summary

Grouping objects by category and aggregating numerical values are one of the most common tasks performed by enterprise-scale software. Relational data generally is better suited to reporting operations than graphs of entity objects, but you'll probably find that you'll need to use LINQ's `Count()`, `Sum()`, `Avg()`, `Min()`, and `Max()` aggregate functions by themselves or in conjunction with the `group join`/`Group Join` operators under a variety of circumstances. Thus, the first part of this chapter, which is about advanced query composition, is devoted to reporting and aggregation, primarily with LINQ expressions.

LINQ handles narrowing filter constraints with `where... && ...` or `Where... And...` expressions but doesn't support widening syntax with `where... || ...` or `Where... Or...` expressions. Additionally, there's no obvious LINQ operator to substitute for SQL's `IN(ArgumentList)` function. LINQ comes to the rescue with the `Contains(Sequence)` operator, which handles both problems, as demonstrated near the middle of the chapter.

It's unrealistic to expect LINQ queries to outperform `SqlDataReaders` or direct connections to alternative data sources. The time required to translate query expressions to method call syntax and "walking" expression trees to generate complex T-SQL, Entity SQL or CAML command strings increases

## Part II: Introducing Language Integrated Query

---

rapidly as queries grow more complex. You can minimize LINQ's performance hit where expression trees are involved by parameterizing and compiling queries that your applications use multiple times. The "Compiling Query Expression Trees to Improve Performance" sample code demonstrates that you can substitute LINQ to SQL for `SqlDataReader` code and only increase execution time of a typical query by about 20 percent.

Finally, creating mock objects for test-driven development is an important requirement to ensure separation of concerns — specifically object manipulation and persistence — when writing unit tests for LINQ applications that involve relational persistence stores or nontraditional data, such as SharePoint lists or data retrieved "from the cloud" with ASP.NET Data Services and LINQ to REST. Therefore, the last part of this chapter is devoted to techniques for generating complete mock object graphs with an in-memory utility (LIMOG) and LINQ-based test harnesses to speed unit tests with complex business objects with complex relationships.

# **Part III: Applying Domain-Specific LINQ Implementations**

**Chapter 5:** Using LINQ to SQL and the LinqDataSource

**Chapter 6:** Querying DataTables with LINQ to DataSet

**Chapter 7:** Manipulating Documents with LINQ to XML

**Chapter 8:** Exploring Third-Party and Emerging LINQ  
Implementations

## Part III: Applying Domain-Specific LINQ Implementations

---

LINQ's architecture supports customization and extensibility for domain-specific language (DSL) implementations. The capability to employ a common set of Standard Query Operators (SQOs) across multiple data domains is one of LINQ's best selling points. Part III's first three chapters cover the following three Microsoft domain-specific LINQ components, which are incorporated into the .NET Framework 3.5 and its extensions:

- ❑ **LINQ to SQL** is an object/relational mapping (O/RM) and object persistence tool for SQL Server 200x. The System.Data.Linq.dll library contains the LINQ to SQL-specific classes. LINQ to SQL's graphical O/RM Designer and its LINQ to SQL Classes template are parts of VS 2008, not .NET 3.5. Chapter 5, "Using LINQ to SQL and the LinqDataSource," covers LINQ to SQL and its dedicated ASP.NET LinqDataSource server control.
- ❑ **LINQ to DataSet** is a domain-specific implementation designed to query and manipulate ADO.NET DataSet objects. Classes in the System.Data.DataSetExtensions.dll library support LINQ to DataSets. Chapter 4, "Querying DataTables with LINQ to DataSets," shows you how to get the most out of traditional ADO.NET DataSet instances with LINQ to DataSet queries.
- ❑ **LINQ to XML** is an Application Programming Interface (API) that's uniquely suited to generating and querying data in XML Infosets. The VB version enables generating XML documents declaratively. LINQ to XML represents a potential substitute for XPath, XSLT, XQuery, or all three, for querying and transforming XML data. System.Linq.Xml.dll provides LINQ to XML's special classes. Silverlight 2 supports LINQ to XML in order to simplify Asynchronous JavaScript and XML (AJAX) programming with C# and VB.

Part III's fourth chapter, Chapter 8, "Exploring Third-Party and Emerging LINQ Implementations," includes sample C# and VB projects to demonstrate the capability of the following third-party and emerging Microsoft domain-specific LINQ providers:

- ❑ **LINQ to Active Directory** (LINQ to AD, formerly LINQ to LDAP) was one of the first full-featured domain-specific LINQ implementations to be written by a third-party developer, Bart De Smet. Active Directory's query language is based on the arcane Lightweight Directory Access Protocol (LDAP) filter syntax; LINQ to AD translates LINQ expressions and SQO method calls to LDAP filters that are compatible with the COM-based Active Directory Services Interface (ADSI) and .NET's System.DirectoryServices classes.
- ❑ **LINQ to SharePoint** enables querying Windows SharePoint Services (WSS) 2+ and Windows SharePoint Server lists by translating LINQ expressions and SQO method calls to Collaborative Application Markup Language (CAML) by means of method calls to the SharePoint's Lists Web service or direct operations with the SharePoint object model. LINQ to SharePoint was the last LINQ project that De Smet completed before he went to work for Microsoft in late 2007 and has much in common with the LINQ to SQL's application design.
- ❑ **LINQ to REST** (Representational State Transfer) is a component of Microsoft's ADO.NET Data Services (ANDS, formerly Project Astoria), which was in the development stage when this book was written. ANDS is designed to enable querying any data resource that supports the IQueryble interface, such as LINQ to SQL or LINQ to Entities, with a query language based on Universal Resource Indicators and the HTTP GET method that you can type in a browser's address box. ANDS returns query results in Atom 1.0 feed and JavaScript Object Notation (JSON) formats. ANDS is intended as a data source for SilverLight projects and Web service mashups, such as those that Microsoft's PopFly application enables. LINQ to REST translates LINQ expressions and SQO method calls to the URI query syntax; data updates use the Atom Publishing Protocol over HTTP.

## Part III: Applying Domain-Specific LINQ Implementations

---

- ❑ **Parallel LINQ (PLINQ)** is an API for parallel processing of LINQ queries on PCs and servers that have multi-core processors. As processor speeds increase, so do operating power requirements. With reduced power consumption by desktop and racked servers, as well as longer battery life for laptops and other mobile PC form factors becoming increasingly important to potential purchasers, multi-core processors are quickly becoming more popular. To obtain better performance with the same or slower clock rates, applications must be capable of dividing the processing workload between cores. PLINQ's objective is to simplify programming of parallel processing for LINQ queries against in-memory objects.

The following three sections provide more details about the three domain-specific LINQ implementations that ship with all VS 2008 editions, with emphasis on LINQ to SQL because of its early popularity.

## LINQ to SQL

It's a reasonably safe assumption that 90 percent or more of all commercial .NET projects involve retrieving information persisted in tables of relational database management systems (RDBMSs). As an increasing percentage of software architects and developers have adopted object-oriented (OO) programming techniques, databases have become the preferred storage mechanism for persisting and sharing business objects. Databases designed specifically to store and retrieve objects — called *object-oriented database management systems (OODBMSs)* — model and create data as objects and support classes, properties, and methods, as well as inheritance by subclasses. Despite the natural affinity between business objects and OODBMSs, the ubiquity, performance, and economics of relational databases have resulted in their domination of the business object persistence market. This incongruity caused the emergence of two classes of developers: those whose data-centric *weltanschauung* is rows and columns of table data linked by relationships and others whose OO view of the world consists of sets of object graphs, properties, methods, and associations. Virtually all Java programmers and the vast majority of .NET developers fall in the latter, OO category. Database administrators (DBAs) increasingly dominate the data-centric clan.

Use of an RDBMS as an object persistence framework implies the use of SQL as the programming language for CRUD (create, retrieve, update, and delete) operations. About 10 years before VS 2008's release, Microsoft's SQL Server group started a development project for an O/RM tool for object persistence management with SQL Server. Microsoft called the fledgling O/RM tool *ObjectSpaces* and code-named it "Orca" during its early, pre-alpha stage. The initial ObjectSpaces team consisted of Luca Bolognese, program manager, and Matt Warren, lead (and only) developer. An early ObjectSpaces beta version appeared in Visual Studio 2005 (Whidbey) Beta 1 and was removed from Beta 2. The black hole of WinFS — short for *Windows Future Storage* — the ill-fated file system planned for "Longhorn Client," swallowed ObjectSpaces. "Longhorn Client" became Windows Vista, and "Longhorn Server" was the code name for Windows Server 2008. Microsoft abandoned WinFS late in the Windows Vista beta period.

*One of the items on the wish list for ObjectSpaces was to replace brittle, string-based SQL queries with a domain-specific query language that the compilers for Windows programming languages could understand. At the time, XML was the "next big thing" for programming so OPath — ObjectSpaces' query language — queries were strings of an XPath dialect. So ObjectSpaces simply replaced brittle SQL strings with brittle OPath strings.*

## Part III: Applying Domain-Specific LINQ Implementations

---

Lack of a Microsoft O/RM created a market opportunity for independent software vendors (ISVs) and individual developers alike. By the time LINQ to SQL appeared as DLinq in the May 2006 LINQ Community Technical Preview (CTP) that worked with VS 2005, there were more than 40 commercial and open-source O/RM implementations for .NET 2.0 and earlier. Many of these products became quite successful. For example, NHibernate — a open-source .NET derivative of Java's Hibernate persistence framework — gained many adherents among .NET Most Valued Professionals (MVPs); LLBLGen Pro and the Wilson ObjectMapper gained the lead among commercial O/RM products.

LINQ to SQL offers a relatively lightweight, high-performance object persistence platform with an easy-to-use graphical mapping tool that supports stored procedures for creating, retrieving, updating, and deleting business objects. LINQ to SQL also offers optimistic concurrency conflict management by providing original, current, and database values of conflicting updates. ASP.NET 3.5 includes a LinqDataSource server control specifically designed to take advantage of LINQ to SQL's feature set.

LINQ to SQL's primary limitations are:

- ❑ It supports only SQL Server 200x, SQL Server Express and SQL Server Compact Edition 3.5 as the back-end data store because its expression tree limits its output SQL Server's T-SQL dialect. Its competitors offer the option of support for most commercial and popular open-source data stores. The Entity Framework, which is the subject of this book's last two parts, enables database vendors and third-party data provider suppliers to take full advantage of the Entity Data Model and LINQ to Entities.
- ❑ It's intended for projects where there's a one-to-one relationship between database tables and business entities. A partial exception is support for the table-per-hierarchy (TPH) inheritance model in which a single table can implement entities that identify subclasses by values in a discriminator column. The Entity Data Model supports TPH, table-per-type (TPT), and table-per-concrete-type (TPCT) hierarchies.
- ❑ Its top-level `DataContext` object isn't serializable, so it's not feasible to write data layers on separate physical tiers that support object tracking and optimistic concurrency management. Future versions might overcome this limitation with a lightweight, connectionless `DataContext` that supports `DataContract` serialization offered by .NET 3.x and Windows Communication Framework (WCF). An unsupported set of `EntityBag` and related classes enable serializing an `DataContext` proxy to implement a three-tier Entity Framework architecture.

Despite the preceding restrictions, LINQ to SQL probably will be the most popular of the three Microsoft domain-specific LINQ implementations that shipped with VS 2008.

*In what might be a unique experience among Microsoft employees, Luca Bolognese (<http://blogs.msdn.com/lucab01>) and Matt Warren (<http://blogs.msdn.com/mattwar>) survived the demise of ObjectSpaces, and four years later remain a lead program manager and principal architect, respectively, of LINQ and more particularly LINQ to SQL. Warren published a nine-part tutorial for writing expression trees and related classes as the "LINQ: Building an IQueryable Provider" series (<http://blogs.msdn.com/mattwar/search.aspx?q=iqueryable&p=1>).*

### LINQ to DataSet

DataSets have been ADO.NET's favored approach to processing relational data and binding tabular data structures to WinForm and WebForm controls, such as grids, lists, and textboxes. LINQ to DataSet enables writing LINQ queries to filter and sort typed DataTables without the need to define DataViews by using SQL-like, string-based expressions. DataSets don't support joins between DataTables, but LINQ to DataSet enables emulating joins between DataTables with `group...by...into/Group...By...Into` expressions and the `GroupBy` SQO. Grouping operations with LINQ to DataSet can return flat or hierarchical representations of joined DataTables, and apply aggregate operators to compute average, minimum, maximum or total values for designated groups and entire collections.

### LINQ to XML

Querying and transforming XML data in the form of XML Infosets traditionally has required a combination of XSLT transforms and XPath expressions or, more recently, XQuery expressions. The need to learn two or three languages just to query or transform XML data appears to many developers as a hurdle that leads them back to writing clumsy, imperative, sequential XML Document Object Model (DOM) code that reduces performance and consumes large amounts of memory. This is especially true as VB programmers moving from COM-based VB6 and earlier to VB .NET and adopting a more object-oriented worldview. The goal of the LINQ to XML API, which was Microsoft's first domain-specific LINQ implementation, was to minimize the Object-to-XML (O/X) "impedance mismatch." Erik Meijer, called by most the "father of LINQ," devoted much of his initial LINQ design efforts to turning *triangles*, which symbolize hierarchical XML trees, into *circles*, the most common graphic representation of objects, and vice versa. His efforts were remarkably successful.

LINQ to XML treats XML Infosets as hierarchical collections of XElement objects, which may have optional `xAttribute` name/value pairs. Adding an XML Declaration, processing instruction, or other document-related component requires enclosing the XElement s in an XDocument object. However, LINQ to XML handles document fragments; the only requirement is that fragments must be well formed. The API includes a lightweight and performant replacement for the XML DOM that you can query with familiar SQOs and transform them to other XML documents or return scalar values, such as aggregates. LINQ to XML provides C# programmers with a simplified method of programmatically authoring XML Infosets with a process called *function construction*. For a change, VB programmers get more sophisticated XML document authoring capabilities than their C# counterparts: VB 9.0's XML literals feature makes XML a first-class component of the language itself.



# 5

## Using LINQ to SQL and the LinqDataSource

LINQ to SQL is one of the three domain-specific LINQ implementations included in the .NET Framework 3.5 upgrade and VS 2008 RTM. Chapters 6, “Querying DataTables with LINQ to DataSets,” and Chapter 7, “Manipulating Documents with LINQ to XML,” cover the other two. LINQ to SQL began life in 2003 as a project intended to become the successor to ObjectSpaces, Microsoft’s first serious attempt to deliver a commercial object/relational mapping (O/RM) tool. O/RM is intended to simplify the process of exchanging data created by object-oriented programs with relational databases. Microsoft initially intended Object Spaces to become part of Visual Studio 2003 (Everett) and later included it in an early beta version of VS 2005. The ill-fated Windows Future Storage (WinFS) relational database for what was to become the Windows Vista operating system adopted Object Spaces as the query engine for its client API. When Microsoft abandoned WinFS as a product in June 2006, ObjectSpaces died with it. Early VS 2008 beta versions included DatabaseLINQ (DLinq) and XmlLINQ (XLinq), which the Visual Studio team renamed LINQ to SQL and LINQ to XML two days before the announcement of WinFS’s demise.

O/RM tools for .NET is a highly competitive market. There were more than 60 commercial and open-source O/RM tools listed in the SharpToolbox’s Object-Relational Mappers category (<http://sharptoolbox.com/categories/object-relational-mappers>) when this book was written. Most O/RM tools offer two basic features: object persistence and an object-oriented query language.

*Object persistence* is the process of saving the state of in-memory object instances, called *entities*, to tables of a relational database; persistence implements create, update, and delete (CUD) operations. LINQ to SQL’s top-level object, `DataContext`, has a generic `Table< TEntity >` collection for each entity. These collections cache state information — such as whether an entity has been added, updated, or deleted — and manage retrieving (hydrating) and persisting (dehydrating) objects. The `DataContext` also implements an `IdentityManager` to ensure that only a single copy of an individual entity resides in memory and a `ChangeTracker` to keep state information up to date.

## Part III: Applying Domain-Specific LINQ Implementations

---

*Object-oriented query languages* (OOQLs) retrieve instances from the persistence store to complete the CRUD acronym. Most OOQLs resemble SQL but have extended syntax to accommodate the lack of database-style foreign-key relationships between objects and dependent subobjects. LINQ to SQL is unique among O/RMs because it uses LINQ as its OOQL. LINQ to SQL replaces traditional SQL string-based queries, which most developers consider *brittle* (because they break easily), with compiler-validated, IntelliSense-enabled, strongly typed LINQ expressions. LINQ to SQL makes extensive use of expression trees to translate LINQ query expressions to SQL statements for execution by the persistence store. The data source for all LINQ to SQL query expressions is one or more `Table< TEntity >` collections.

Prof. Peter Chen, who first proposed the Entity-Relationship Model (E-RM) for data in 1976, describes entities and relationships as: “An *entity* is a ‘thing,’ which can be distinctly identified. A specific person, company, or event is an example of an entity. A *relationship* is an association among entities. For instance ‘father-son’ is a relationship between two ‘person’ entities.” To avoid confusion with relationships of relational databases, this book uses the term *association* between objects and related objects. LINQ to SQL calls one:many ( $1:n$ ) associations, such as `Customers.Orders`, `EntitySets` and many:one ( $m:1$ ) associations (`Order.Customer`) `EntityRefs`. LINQ to SQL doesn’t support many:many ( $m:n$ ) associations (`Products:Orders`) directly; an *association entity*, which is equivalent to the E-RM’s *relationship record*, is required. Association entities can have an optional *payload*, such as `Order_Detail.Quantity` and `Order_Detail.Price`.

*LINQ to SQL autogenerateds CLR entity classes with singular names from relational tables, which usually have plural names. LINQ to SQL enables its singularizer (also called a de-pluralizer), which is generally effective, for English locales. A pluralizer autogenerateds plural names for `DataContext.Table< TEntity >` objects from the class names. Singularization and pluralization are turned off by default in VS 2008 versions for non-English-speaking locales. You can prevent singularization with English VS 2008 versions by opening the Options dialog’s Database Tools | O/R Designer page and setting the Pluralization of Names’ Enabled property value to False.*

O/RM tools differ widely in the ease of use and flexibility of their methods for mapping objects to relational tables (or vice versa) and their support for various relational database management systems. As mentioned later in the chapter, LINQ to SQL supports only table-per-hierarchy inheritance mapping; most other O/RMs also handle table-per-type and table-per-concrete-type mapping. `SqlClient` is the only data provider for LINQ to SQL, so the implementation only supports SQL Server 200x [Express] and SQL Compact (SSCE) 3.5. On the other hand, VS 2008’s graphical O/R Designer for LINQ to SQL makes creating SQL Server 200x object data sources for Windows or Web form clients as quick and easy as generating typed DataSets. As you’ll see in this chapter’s “ASP.NET Databinding with the `LinqDataSource` Control” section, ASP.NET 3.5’s `LinqDataSource` control connects directly to a LINQ to SQL `DataContext` object and `GridView` control to enable server-side paging and sorting with almost no code behind the `*.aspx` file.

*The Designer doesn’t support SSCE 3.5, so you must use the `SqlMetal.exe` command-line tool to generate data sources. This chapter’s “Generating Partial Entity Classes and Mapping Files with `SqlMetal.exe`” section describes how to use the command-line tool.*

# Object/Relational Mapping with LINQ to SQL

LINQ to SQL is more than just a LINQ implementation for relational databases; it includes an easy-to-use graphical object/relational mapping (O/RM) tool. The O/RM tool generates an entity class for each table you select from the SQL Server 200x database to which the project connects. The tool also generates an *association* for each relationship between the underlying tables. Associations eliminate the need to specify joins in LINQ queries that include projections of fields from related tables. The ability to navigate object associations is an important feature of LINQ implementations and the Entity Framework.

The advantage of moving from the relational model of tables, rows, and keys to the entity model of entity classes, entity instances, and associations is that entities can — and usually do — represent business objects, such as customers, partners, vendors, employees, products, or services. If your databases are highly [over]normalized, such as the AdventureWorks sample database, you'll probably need one or more SQL joins and WHERE clause criteria to represent a customer or employee fully. Adopting an entity model enables developers to design real-world objects with custom properties, associations, and behaviors suited to the organization's business processes. LINQ to SQL delivers the rapid application development (RAD) methods that make moving from the relational to the entity model practical for small to medium-sized projects.

*LINQ to SQL is intended for 1:1 mapping of tables:entities, although it does have limited support for table-per-hierarchy entity inheritance based on discriminator columns to identify subclasses. Read-only entity sets can be based on views or stored procedures. The ADO.NET Entity Framework's Entity Data Model supports joining multiple tables to create a single updatable entity set, as well as two additional types of entity inheritance strategies. The Entity Framework is directed at medium- to large-scale projects. The ADO.NET Team released the Entity Framework (EF) and Entity Data Model (EDM) in VS 2008 SP1 in August 2008.*

## Mapping Tables to Entity Sets with the LINQ to SQL O/R Designer

LINQ to SQL Classes is the name of a new template for C# and VB projects (see Figure 5-1). Adding a new LINQ to SQL class to a Web site project directly opens a message box that asks if you want to add the LINQ to SQL files to your site's App\_Code folder. Clicking OK or adding the LINQ to SQL class to a class library opens an empty O/R Designer surface for the *DatabaseName.dbml* mapping layer file, *Northwind.dbml* for this example, and a reference to *System.Linq.Data.dll*. Alternatively, you can drag graphical Class, Association and Inheritance tools from the designer's Toolbox to create custom

## Part III: Applying Domain-Specific LINQ Implementations

business object classes and associations, and then add members to the class widgets to start from scratch with a domain-driven design. When you complete the custom design, invoking the `DataContext.CreateDatabase()` method generates the SQL Server 200x database, tables, and foreign-key constraints to persist the corresponding objects and associations.

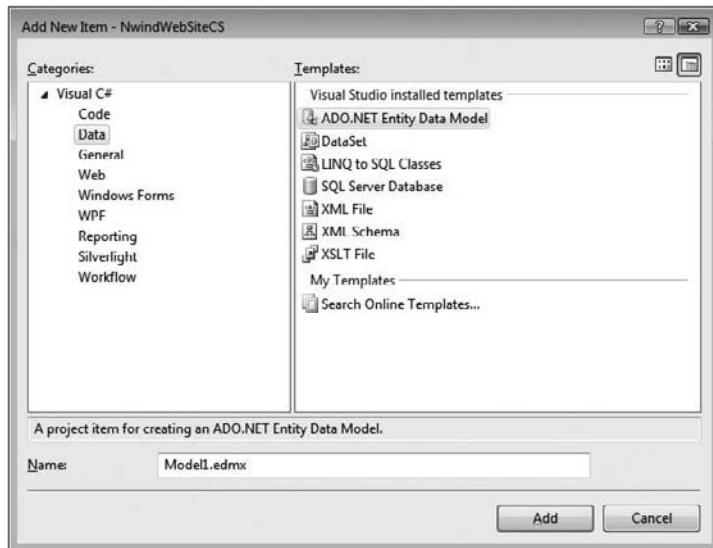


Figure 5-1

It's a much more common practice, at least among .NET developers, to start with metadata from an existing database, but this practice is frowned upon by domain-driven design (DDD) and test-driven development (TDD) advocates. Dragging table nodes from the Server Explorer (Database Explorer in the Visual Web Developer 2008 Express version) to the designer surface adds class widgets to represent autogenerated entities and connection lines for associations, as shown in Figure 5-2.

## Chapter 5: Using LINQ to SQL and the LinqDataSource

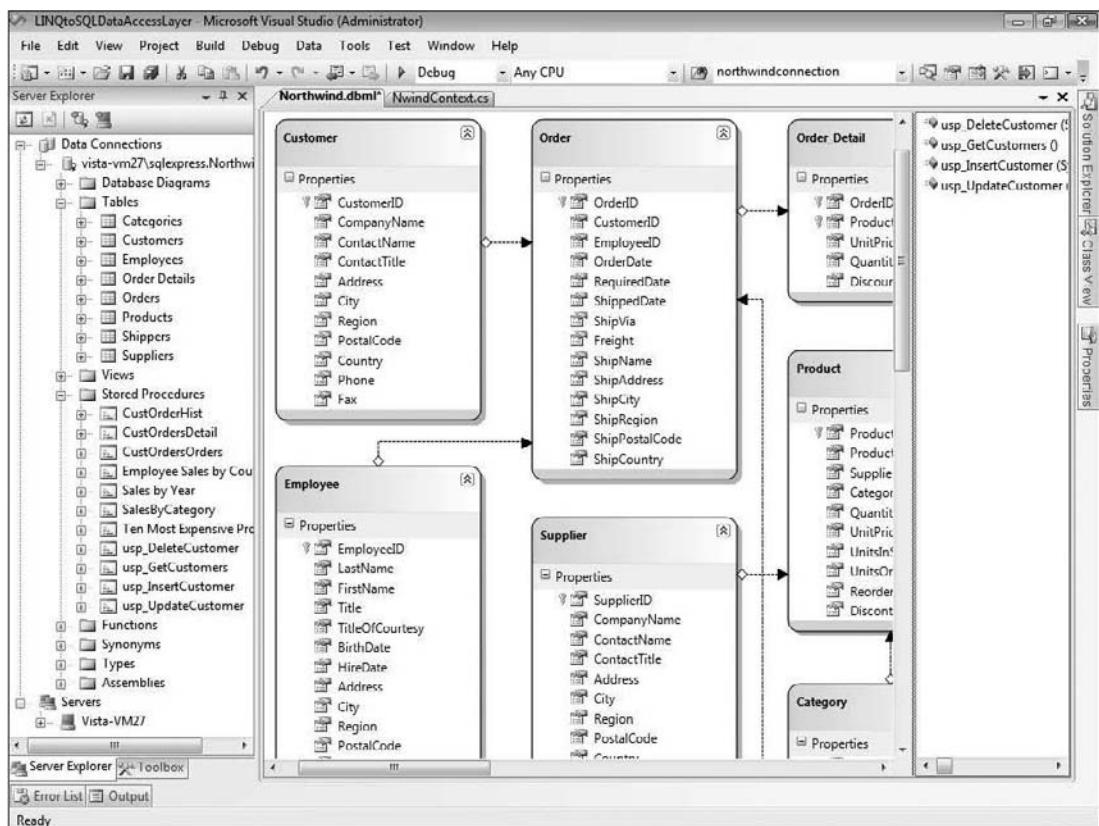


Figure 5-2

The O/R Designer provides property sheets to customize the `DataContext`, as well as each entity class and its public members (see Figure 5-3). Entity class widgets display member lists and enable editing property names. For example, you can change the entity's `Name` property value; when you do, the entity collection's name changes to the English plural of the new entity name. The entity class property sheet also has `Delete`, `Insert`, and `Update` properties to let you use the `Configure Behavior` dialog to replace the default value (Use Runtime for dynamically generated T-SQL) with the name of a user stored procedure for table updates. The later "Substituting Stored Procedures for Dynamic SQL" section provides further details on the benefits and limitations of stored procedures with LINQ to SQL.

## Part III: Applying Domain-Specific LINQ Implementations

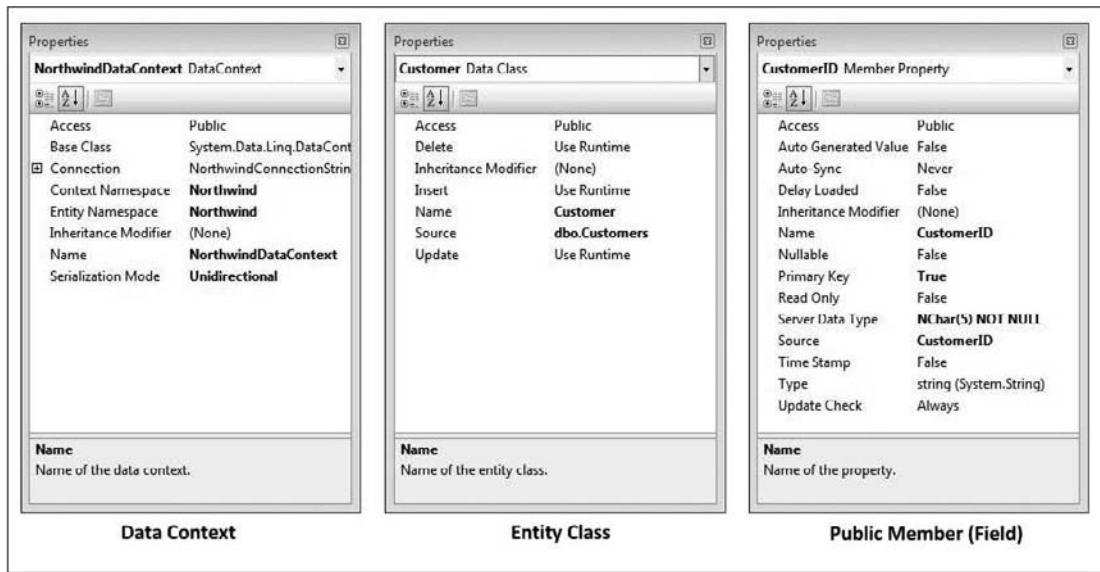


Figure 5-3

*The O/R Designer requires building the project to generate the code for the underlying classes by the designer before you refer to the classes in programming code. Alternatively, you can right-click the `FileName.dbml` file in Solution Explorer and choose Run Custom Tool to generate the classes.*

## Generating Partial Entity Classes and Mapping Files with SqlMetal.exe

LINQ to SQL also offers the option of using a command-line class and XML file creation tool called `SqlMetal.exe`, which the O/R Designer uses to generate its attribute-based object/relational mapping classes. You can use `SqlMetal.exe` to generate:

- Source code for partial entity classes with mapping attributes from a SQL Server 200x [Express] or SQL Server Compact (SSCE) v.3.5 database
- Source code for partial entity classes with added code to support SQL Server stored procedures, views and table-valued functions
- An O/R Designer (\*.dbml) file from a SQL Server or SSCE v.3.5 database
- Source code for partial entity classes with mapping attributes from an O/R Designer (\*.dbml) file
- An \*.xml mapping file and source code for partial entity classes without mapping attributes from a SQL Server, SSCE v.3.5 database or \*.dbml file

Here's the command-line specification: `SqlMetal[.exe] [options] [input file]`

## Chapter 5: Using LINQ to SQL and the LinqDataSource

---

`SqlMetal.exe` is located in the `\Windows\Microsoft .NET\Framework\v3.5` folder, so it's on the path that the Visual Studio 2008 Command Prompt adds to the environment. Typing **SqlMetal** at the command prompt returns usage information.

The following table lists command-line options.

Option	Description
<code>/server:name</code>	Database server name (use [input file] for .mdf or .sdf files)
<code>/database:name</code>	Database catalog on the server
<code>/user:name</code>	SQL Server login ID (default is Windows Authentication)
<code>/password:password</code>	SQL Server password (default is Windows Authentication)
<code>/conn:connection string</code>	SqlClient database connection string (don't use with the <code>/server</code> , <code>/database</code> , <code>/user</code> , or <code>/password</code> options)
<code>/timeout:seconds</code>	Timeout value for database connection (default is 0 =∞)
<code>/sprocs</code>	Extract all stored procedures along with all tables
<code>/views</code>	Extract all database views along with all tables
<code>/functions</code>	Extract all database functions along with all tables
<code>/dbml[:filename.dbml]</code>	Output as *.dbml file (can't be used with the <code>/map</code> option)
<code>/code[:filename.cs   :filename.vb]</code>	Output as class source code (don't use with the <code>/dbml</code> option)
<code>/map[:filename.xml]</code>	Generate XML mapping file to eliminate attributes (don't use with the <code>/dbml</code> option)
<code>/language:language</code>	Class source code language (default is cs or derived from extension on code file name)
<code>/namespace:name</code>	Namespace of generated code (default is none)
<code>/context:typename</code>	Name of <code>DataContext</code> class (default is <code>DatabaseName</code> , <code>DatabaseNameDataContext</code> is recommended)
<code>/entitybase:typename</code>	Base class of entity classes in the generated code (default is no base class)
<code>/pluralize</code>	Automatically pluralize or de-pluralize class and member names employing rules for the English language (default is no pluralization)
<code>/serialization:option</code>	Generate serializable classes for WCF: none or unidirectional (default is None; bidirectional serialization isn't available)
<code>/provider:typename</code>	Provider type (default is autodetection of provider at run time)

## Part III: Applying Domain-Specific LINQ Implementations

---

input file can be a SQL Server 200x [Express] \*.mdf file, a SQL Server CE 3.5 \*.sdf file, or a \*.dbml O/R Designer file.

Following are a few observations about the behavior of `SqlMetal.exe`:

- ❑ `SqlMetal.exe` generates partial entity classes for all tables, as well as stored procedures, table-valued inline and multi-statement functions, views, and any combination if specified by the appropriate option(s). There is no means to specify a list of database objects for `SqlMetal.exe` to process or not to process.
- ❑ If you don't supply a filename with the /dbml, /code, or /map options, what would be the file's content streams to the command prompt.
- ❑ `SqlMetal.exe` doesn't add appropriate .cs/.vb, .dbml, or .xml extensions to filenames if they're missing.
- ❑ The /map option requires *Filename.cs* or *Filename.vb*, and *Filename.xml* filespecs to write the required pair of code and mapping files.
- ❑ `SqlMetal.exe` writes files to the command prompt's current folder
- ❑ Prefixing *filename* with the ~%dp0 macro substitutes for itself the well-formed path to the folder containing the batch file that executes `SqlMetal.exe` instructions.

Here's a typical command line instruction to generate a VB 9.0 partial class file from Northwind running on an SQL Server 200x Express instance:

```
SqlMetal /server:.\SQLEXPRESS /database:Northwind /code:Northwind.vb  
/context:NorthwindDataContext /pluralize
```

This instruction generates a C# 3.0 partial class file and XML mapping file from Northwind dynamically attached to a SQL Server 200x Express instance:

```
SqlMetal /map:\mappath\Northwind.xml /code:\codepath\Northwind.cs  
/namespace:Northwind /context:NorthwindDataContext /pluralize  
\datapath\Northwind.mdf
```

Use this instruction to generate a `Northwind.dbml` O/R Designer file (also called an intermediate file) from a SQL Server Compact Edition v.3.5 database:

```
SqlMetal /dbml:Northwind.dbml Northwind.sdf
```

Here's an instruction to generate a serializable C# 3.0 partial class file and a companion XML mapping file from an O/R Designer (intermediate) file:

```
SqlMetal /map:\mappath\Northwind.xml /code:\codepath\Northwind.cs  
/namespace:Northwind /context:NorthwindDataContext /pluralize  
/serialize:unidirectional \mappath\Northwind.dbml
```

If your source database is SSCE v.3.5 you must use `SqlMetal.exe` to create your partial class file; the O/R Designer doesn't support \*.sdf files. You'll probably find `SqlMetal.exe` to be faster than the designer for generating partial classes from databases that have a large number of tables. However,

the O/R Designer enables specifying individual stored procedures for CRUD operations and lets you customize mapping without manually editing the \*.dbml or companion \*.xml file.

### **Working with \*.dbml and \*.xml Mapping Files**

The \*.dbml (database mapping language) and \*.xml mapping files that SqlMetal.exe generates are similar in structure and XML content. For example, both file types displays a `<Table>` element and `<Type>` subelement for each table in the database and share many common attribute name-value pairs. However, the two are used for entirely different purposes. The following sections describe the content and editing techniques for these two file types.

#### **Editing \*.dbml Files in the Designer**

The \*.dbml files produced by SqlMetal.exe or the O/R Designer map each relational `<Table>` to a CLR `<Type>` defined by an entity partial class. The following abbreviated example shows the `<Table>` element for the Categories table and part of the `<Table>` element for the Products table. The Member attribute value defines the name of the `DataContext.Table< TEntity >` collection. Most table attributes are named intuitively, but `UpdateCheck` is an exception; its value determines how the column participates in value-based concurrency conflict management. The later “Detecting Concurrency Conflicts” section has more details on this topic.

```
<?xml version="1.0" encoding="utf-16"?>
<Database Name="Northwind"
  xmlns="http://schemas.microsoft.com/linqtosql/dbml/2007">
  <Table Name="dbo.Categories" Member="Categories">
    <Type Name="Category">
      <Column Name="CategoryID" Type="System.Int32" DbType="Int NOT NULL IDENTITY"
        IsPrimaryKey="true" IsDbGenerated="true" CanBeNull="false" />
      <Column Name="CategoryName" Type="System.String" DbType="NVarChar(15)
        NOT NULL" CanBeNull="false" />
      <Column Name="Description" Type="System.String" DbType="NText"
        CanBeNull="true" UpdateCheck="Never" />
      <Column Name="Picture" Type="System.Data.Linq.Binary" DbType="Image"
        CanBeNull="true" UpdateCheck="Never" />
      <Association Name="FK_Products_Categories" Member="Products"
        OtherKey="CategoryID" Type="Products" DeleteRule="NO ACTION" />
    </Type>
  </Table>
  ...
  <Table Name="dbo.Products" Member="Products">
    <Type Name="Product">
      <Column Name="ProductID" Type="System.Int32" DbType="Int NOT NULL IDENTITY"
        IsPrimaryKey="true" IsDbGenerated="true" CanBeNull="false" />
      <Column Name="ProductName" Type="System.String" DbType="NVarChar(40)
        NOT NULL" CanBeNull="false" />
      <Column Name="SupplierID" Type="System.Int32" DbType="Int"
        CanBeNull="true" />
      <Column Name="CategoryID" Type="System.Int32" DbType="Int"
        CanBeNull="true" />
```

## Part III: Applying Domain-Specific LINQ Implementations

---

```
...
<Column Name="Discontinued" Type="System.Boolean" DbType="Bit NOT NULL"
  CanBeNull="false" />
<Association Name="FK_Order_Details_Products" Member="OrderDetails"
  OtherKey="ProductID" Type="OrderDetails" DeleteRule="NO ACTION" />
<Association Name="FK_Products_Categories" Member="Categories"
  ThisKey="CategoryID" Type="Categories" IsForeignKey="true" />
<Association Name="FK_Products_Suppliers" Member="Suppliers"
  ThisKey="SupplierID" Type="Suppliers" IsForeignKey="true" />
</Type>
</Table>
</Database>
```

Each foreign key in a table adds an `<Association>` element to its `<Type>` node.  $1:n$  associations (`EntitySets`), such as `Product.Order_Details` have `OtherKey` and `DeleteRule` attributes. `DeleteRule` specifies one of the SQL `ON DELETE` clause's four allowable actions for dependent records in the `EntityType`: `CASCADE`, `SET DEFAULT`, `SET NULL` or `NO ACTION`. The `ThisKey` and `IsForeignKey="true"` attribute identifies  $m:1$  associations (`EntityRefs`).

The “Generator Tool DBML Reference” section of the March 2007 “LINQ to SQL: .NET Language-Integrated Query for Relational Data” whitepaper (<http://msdn2.microsoft.com/en-us/library/bb425822.aspx>) has a set of tables that fully describe the purpose of all `*.dbml` elements and attributes.

The XML schema file for `*.dbml` files is `\Program Files\Microsoft Visual Studio 9.0\Xml\Schemas\\DbmlSchema.xsd`. `*.xsd` files in this folder appear as items in the XML Schemas list that opens when you choose Schemas from VS 2008’s XML menu. If the schema isn’t selected for you, look for the Target Namespace or File Name columns for the match. Attaching the schema enables IntelliSense for the `*.dbml` file.

When you open a `*.dbml` file in VS 2008 by selecting a large number of tables and dragging them as a group to the empty designer surface, it opens the O/R Designer (refer to Figure 5-2) with table widgets stacked in the upper-left corner of the Design pane. After you rearrange the widgets, saving the design changes generates a `*.dbml.layout` file that contains O/R Designer widget text, size, and location information. Adding the `*.dbml` file to an existing project incorporates the `*.dbml.layout` file and adds a `*.dbml.designer.cs` or `.vb` file. Compiling the project generates the source code for the `DataContext` in the `.cs` or `.vb` file, `Northwind.dbml.designer.cs` for this example. The `*.dbml` file reflects changes you make to `DataContext`, entity, and public member property values in the property sheets (refer to Figure 5-3). Opening the dropdown list of the `DataContext`’s empty `ConnectionString` property value lets you autogenerate a connection string from databases with nodes in Server Explorer and save it in the `app.config` or `web.config` file.

The `*.dbml.designer.cs` file — and thus the assembly — incorporates the mapping data included in the `*.dbml` file as attributes of entities and their members, as described in the chapter’s later “Examining the Generated Classes” section. Therefore, applications or class libraries only need to deploy the `App.config` or `Web.config` file, which contains the connection string.

### **Editing \*.xml Mapping Files in an XML Editor**

Many .NET developers want to free their entity classes of persistence-related clutter — specifically state information, original member values, and mapping attributes. Some commercial O/RM tools generate classes with all three decorations. The `DataContext` object eliminates issues with state and original values, but LINQ to SQL's default application of attributes to entities, members, and associations prevents entities from being represented as “plain old CLR objects” (POCOs). The “Examining the Generated Classes” section further discusses this issue in the context of persistence ignorance.

*POCO is a variation of POJO an acronym for “plain old Java objects.”*

Removing the mapping attributes from the entity classes requires substituting an XML mapping file, which your code must load when the Windows form, Web form or class library opens. Therefore, you must deploy the `*.xml` mapping file with your application or library and use the `*.cs` or `*.vb` file that `SqlMetal.exe` generates from the database instance, `*.mdf`, `*.sdf`, or `*.dbml` file when you specify the `/xml:filename` option. Following is the abbreviated XML mapping file, `Northwind.xml`, that corresponds to the preceding `*.dbml` fragment:

```
<?xml version="1.0" encoding="utf-8"?>
<Database Name="Northwind"
  xmlns="http://schemas.microsoft.com/linqtosql/mapping/2007">
  <Table Name="Categories" Member="Categories">
    <Type Name="Northwind.Category">
      <Column Name="CategoryID" Member="CategoryID" Storage="_CategoryID"
        DbType="Int NOT NULL IDENTITY" IsPrimaryKey="true"
        IsDbGenerated="true" AutoSync="OnInsert" />
      <Column Name="CategoryName" Member="CategoryName" Storage="_CategoryName"
        DbType="NVarChar(15) NOT NULL" CanBeNull="false" />
      <Column Name="Description" Member="Description" Storage="_Description"
        DbType="NText" UpdateCheck="Never" />
      <Column Name="Picture" Member="Picture" Storage="_Picture"
        DbType="Image" UpdateCheck="Never" />
      <Association Name="FK_Products_Categories" Member="Products"
        Storage="_Products" ThisKey="CategoryID" OtherKey="CategoryID"
        DeleteRule="NO ACTION" />
    </Type>
  </Table>
  <Table Name="Products" Member="Products">
    <Type Name="Northwind.Product">
      <Column Name="ProductID" Member="ProductID" Storage="_ProductID"
        DbType="Int NOT NULL IDENTITY" IsPrimaryKey="true"
        IsDbGenerated="true" AutoSync="OnInsert" />
      <Column Name="ProductName" Member="ProductName" Storage="_ProductName"
        DbType="NVarChar(40) NOT NULL" CanBeNull="false" />
      <Column Name="SupplierID" Member="SupplierID" Storage="_SupplierID"
        DbType="Int" />
      <Column Name="CategoryID" Member="CategoryID" Storage="_CategoryID"
        DbType="Int" />
      ...
      <Column Name="Discontinued" Member="Discontinued" Storage="_Discontinued"
```

## Part III: Applying Domain-Specific LINQ Implementations

---

```
    DbType="Bit NOT NULL" />
<Association Name="FK_Order_Details_Products" Member="OrderDetails"
  Storage="_OrderDetails" ThisKey="ProductID" OtherKey="ProductID"
  DeleteRule="NO ACTION" />
<Association Name="FK_Products_Categories" Member="Category"
  Storage="_Category" ThisKey="CategoryID" OtherKey="CategoryID"
  IsForeignKey="true" />
<Association Name="FK_Products_Suppliers" Member="Supplier"
  Storage="_Supplier" ThisKey="SupplierID" OtherKey="SupplierID"
  IsForeignKey="true" />
</Type>
</Table>
</Database>
```

Using one or more XML mapping files to autogenerated class files for O/RM tools is a common practice. ObjectSpaces used this approach, and the Entity Framework relies on three XML mapping files to generate the Entity Data model.

When this book was written, there was no detailed specification or XML schema available for \*.xml mapping files. If the LINQ to SQL team ultimately release a schema, you'll be able to add it to your \Program Files\Microsoft Visual Studio 9.0\Xml\Schemas\ folder for use with VS 2008's XML Editor.

To open a mapping file in conjunction with instantiating a new `DataContext`, you must supply an additional `mappingSource` argument value of the `System.Data.Linq.Mapping.MappingSource` type. The "Instantiating the `DataContext` and Its Object Graph" section shows the syntax. The `System.Data.Linq.Mapping.MappingSource` namespace defines all LINQ to SQL mapping attributes.

## Examining the Generated Classes

Building the project generates C# 3.0 or VB 9.0 partial classes in the `DatabaseName.designer.cs` or `.vb` file for the entities and adds methods for performing CRUD (create, retrieve, update, and delete) operations on the underlying tables. The top-level `DataContext` class acts as the parent object for the entity collections — `Customers`, `Orders`, `Order_Details`, `Employees`, `Shippers`, `Products`, `Suppliers`, and `Categories` for this chapter's example — in `System.Data.Linq.Table< TEntity >` the generic types. Here's the code to define the read-only `Order_Details` entity collection:

### C# 3.0

```
public System.Data.Linq.Table<Order_Detail> Order_Details
{
    get {
        return this.GetTable<Order_Detail>();
    }
}
```

### VB 9.0

```
Public ReadOnly Property Order_Details() As System.Data.Linq.Table(Of Order_Detail)
    Get
        Return Me.GetTable(Of Order_Detail)
    End Get
End Property
```

## Chapter 5: Using LINQ to SQL and the LinqDataSource

Autogenerated entity collection and entity class code uses attributes (emphasized in the following two listings) to map the underlying relational table, columns, and foreign-key constraints to the entity, properties, and associations, as shown in the following abbreviated listings for the Order\_Detail entity:

### C# 3.0

```
[Table(Name="dbo.[Order Details]")]
public partial class Order_Detail : INotifyPropertyChanging, INotifyPropertyChanged
{
    private static PropertyChangingEventArgs emptyChangingEventArgs =
        new PropertyChangingEventArgs(String.Empty);

    private int _OrderID;
    private int _ProductID;
    private decimal _UnitPrice;
    private short _Quantity;
    private float _Discount;
    private EntityRef<Order> _Order;

    private EntityRef<Order> _Order;
    private EntityRef<Product> _Product;

    public Order_Detail() {
        this.Initialize();
    }

    [Column(Storage="_OrderID", DbType="Int NOT NULL", IsPrimaryKey=true)]
    public int OrderID {
        get {
            return this._OrderID;
        }
        Set {
            if ((this._OrderID != value)) {
                if (this._Order.HasLoadedOrAssignedValue) {
                    throw new System.Data.Linq.ForeignKeyReferenceAlreadyHasValueException();
                }
                this.OnOrderIDChanging(value);
                this.SendPropertyChanging();
                this._OrderID = value;
                this.SendPropertyChanged("OrderID");
                this.OnOrderIDChanged();
            }
        }
    }
    ...
}

[Association(Name="Order_Order_Detail", Storage="_Order", ThisKey="OrderID",
    IsForeignKey=true)]
public Order Order {
    get {
        return this._Order.Entity;
    }
}
```

## Part III: Applying Domain-Specific LINQ Implementations

---

```
set {
    Order previousValue = this._Order.Entity;
    if (((previousValue != value)
        || (this._Order.HasLoadedOrAssignedValue == false))) {
        this.SendPropertyChanging();
        if ((previousValue != null)) {
            this._Order.Entity = null;
            previousValue.Order_Details.Remove(this);
        }
        this._Order.Entity = value;
        if ((value != null)) {
            value.Order_Details.Add(this);
            this._OrderID = value.OrderID;
        }
        else {
            this._OrderID = default(int);
        }
        this.SendPropertyChanged("Order");
    }
}
...
...
```

*Partial methods classified as Extensibility Method Definitions have been removed from the sample code, which only lists the class's first property and association. Partial methods simplify implementing corresponding event handlers with code in another partial class file that the designer doesn't affect. The "Eager-Loading EntityRef Values to Reduce Database Server Traffic" section near the end of this chapter demonstrates implementation of the DataContext.OnCreate() partial method as an event handler.*

### VB 9.0

```
<Table(Name:="dbo.[Order Details]")> _
Partial Public Class Order_Detail
    Implements System.ComponentModel.INotifyPropertyChanging, _
    System.ComponentModel.INotifyPropertyChanged

    Private Shared emptyChangingEventArgs As PropertyChangingEventArgs = _
        New PropertyChangingEventArgs(String.Empty)

    Private _OrderID As Integer
    Private _ProductID As Integer
    Private _UnitPrice As Decimal
    Private _Quantity As Short
    Private _Discount As Single
    Private _Order As EntityRef(Of [Order])
    Private _Product As EntityRef(Of Product)

    Public Sub New()
        MyBase.New
        OnCreated
        Me._Order = CType(Nothing, EntityRef(Of [Order]))
        Me._Product = CType(Nothing, EntityRef(Of Product))
    End Sub

```

## Chapter 5: Using LINQ to SQL and the LinqDataSource

```
End Sub

<Column(Storage:="_OrderID", DbType:="Int NOT NULL", IsPrimaryKey:=true)> _
Public Property OrderID() As Integer
    Get
        Return Me._OrderID
    End Get
    Set
        If ((Me._OrderID = value) =
            = false) Then
            If Me._Order.HasLoadedOrAssignedValue Then
                Throw New System.Data.Linq. _
                    ForeignKeyReferenceAlreadyHasValueException
            End If
            Me.OnOrderIDChanging(value)
            Me.SendPropertyChanging
            Me._OrderID = value
            Me.SendPropertyChanged("OrderID")
            Me.OnOrderIDChanged
        End If
    End Set
End Property

<Association(Name:="Order_Order_Detail", Storage:="_Order", _
    ThisKey:="OrderID", IsForeignKey:=true)> _
Public Property [Order]() As [Order]
    Get
        Return Me._Order.Entity
    End Get
    Set
        Dim previousValue As [Order] = Me._Order.Entity
        If ((previousValue Is value) = false) =
            OrElse (Me._Order.HasLoadedOrAssignedValue = false)) Then
            Me.SendPropertyChanging
            If ((previousValue Is Nothing) = false) Then
                Me._Order.Entity = Nothing
                previousValue.Order_Details.Remove(Me)
            End If
            Me._Order.Entity = value
            If ((value Is Nothing) = false) Then
                value.Order_Details.Add(Me)
                Me._OrderID = value.OrderID
            Else
                Me._OrderID = CType(Nothing, Integer)
            End If
            Me.SendPropertyChanged("[Order]")
        End If
    End Set
End Property
...
End Class
```

*The Table, Column, and Association attributes and their parameter:value pairs are copied directly from the \*.dbml or \*.xml mapping file, depending on how you create the entity class file.*

## Part III: Applying Domain-Specific LINQ Implementations

---

As mentioned earlier in the “Editing \*.xml Mapping Files in an XML Editor” section, decorating class and property declarations with attributes from the `System.Data.Linq` namespace results in *attribute-centric mapping*, which is also frowned upon by O/RM purists who insist on *persistence ignorance*. *Persistence ignorance* requires the business object layer to be completely unaware of the technique for persisting (saving) objects and to tolerate a change in persistence methodology without code modification. Attributes incorporate table, column, and foreign-key constraints in entity, property, and association declarations. In the event of a change in the underlying relational database schema or database type, you must at least recompile the project, if you use the graphic designer and a \*.dbml file. You can increase persistence ignorance by using an .xml mapping file and editing it to reflect minor schema changes. You’ll still have a *leaky abstraction* because most LINQ to SQL applications require access to the `DataContext`’s `Table< TEntity >` collections as data sources for LINQ query expressions. A *leaky abstraction* is a design in which implementation details “leak through” into the business entity. There are few — if any — O/RM tools that deliver total persistence ignorance.

## **Instantiating the `DataContext` and Its Object Graph**

In contrast to ADO.NET DataSets, which you materialize as a persisted component from VS 2008’s Toolbox, you create instances of the `DataContext` object with code. Alternatively, you can add a `LinqDataSource` or `ObjectDataSource` control to your page and specify the `DataContext` or a `Table< TEntity >` as its `DataSource` property value, respectively; this approach is the subject of the later “ASP.NET Databinding with the `LinqDataSource` Control” section.

Typically, you instantiate the `DataContext` in the `Page_Load` event handler with the following code:

### **C# 3.0**

```
protected void Page_Load(object sender, EventArgs e) {
    if (!IsPostBack) {
        using (NorthwindDataContext dcNwind =
            new NorthwindDataContext(
                Properties.Settings.Default.NorthwindConnectionString)) {
            int customerCount = dcNwind.Customers.Count();
            int orderCount = dcNwind.Orders.Count();
            // ... Queries and related code
        }
    }
}
```

### **VB 9.0**

```
Protected Sub Customers_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load
    If Not IsPostBack Then
        Using dcNwind As New NorthwindDataContext(_
            My.Settings.NorthwindConnectionString)
            Dim intCustomers As Integer = dcNwind.Customers.Count()
            Dim intOrders As Integer = dcNwind.Orders.Count()
            ' ... Queries and related code
        End Using
    End If
End Sub
```

*You don’t need to supply the `connectionString` parameter if your `DataContext` class has a parameterless constructor.*

## Chapter 5: Using LINQ to SQL and the LinqDataSource

If you use a \*.xml mapping file, you must add the emphasized code, which includes the mappingSource parameter, in the following sample to load and use its metadata for mapping:

### C# 3.0

```
protected void Page_Load(object sender, EventArgs e) {
    if (!IsPostBack) {
        XmlMappingSource mappingSource =
            XmlMappingSource.FromXml(File.ReadAllText("d:\path\mapping.xml"));
        using (NorthwindDataContext dcNwind = new NorthwindDataContext(
            Properties.Settings.Default.NorthwindConnectionString,
            mappingSource)) {
            int customerCount = dcNwind.Customers.Count();
            int orderCount = dcNwind.Orders.Count();
            // ... Queries and related code
        }
    }
}
```

### VB 9.0

```
Protected Sub Customers_Load(ByVal sender As Object, _
                             ByVal e As System.EventArgs) Handles Me.Load
    If Not IsPostBack Then
        Dim MappingSource As XmlMappingSource = _
            XmlMappingSource.FromXml(File.ReadAllText("d:\path\mapping.xml"))
        Using dcNwind As New NorthwindDataContext( _
            My.Settings.NorthwindConnectionString, MappingSource)
            Dim intCustomers As Integer = dcNwind.Customers.Count()
            Dim intOrders As Integer = dcNwind.Orders.Count()
            ' ... Queries and related code
        End Using
    End If
End Sub
```

*Declaring dcNwind as a form-level variable to cache the XmlMappingSource will improve the performance of Windows form projects that execute multiple queries.*

The preceding example instantiates an empty (dehydrated) object graph. The two `Count()` method calls translate to these two T-SQL batches, each of which runs on a separate pooled connection and returns a scalar (integer) value:

### T-SQL Batches

```
SELECT COUNT(*) AS [value] FROM [dbo].[Customers] AS [t0]
SELECT COUNT(*) AS [value] FROM [dbo].[Orders] AS [t0]
```

Fortunately, the dehydrated `DataContext` is a relatively lightweight object, so creating a new instance doesn't affect performance significantly. Connection pooling minimizes the effect on performance of opening a new connection for each LINQ query you execute.

*If you execute parameterized stored procedures for all CRUD operations, it's possible to write simple applications that fill editable databound grids without executing any LINQ query expressions.*

# Using LINQ to SQL as a Data Access Layer

Since ADO.NET version 1.0, DataSets have formed the foundation of the .NET Framework's data-access technology. DataSets are an in-memory representation of the source database's tables and relationships that usually contain a subset of the available data. Strongly typed DataSets store the local data as Common Language Runtime (CLR) data types and support two-way, read/write binding to data-aware Windows and Web form controls, such as grids and lists. You can populate a DataSet, disconnect it from the source database, make changes to the data, and then reconnect to update the underlying tables. You can persist the DataSet locally between sessions by serializing it to an XML document or binary stream that you store in the file system; serialized DataSets can traverse tiers by .NET remoting or as XML Web services. (DataSets use an XML schema as their contract.) Optimistic concurrency management provides flexibility for resolving conflicting updates. Although DataSets are instances of .NET classes, they retain the tabular, row-column architecture of the relational world.

A data access layer (DAL) is a software component that abstracts the data model from its underlying physical storage model in the database and encapsulates a standardized set of data query and update methods. A DAL should isolate upper-level layers that implement business logic, data validation, and the user interface from knowledge of and potential interference with the database and its connection. Another DAL objective is to eliminate or minimize the effect of database schema changes on upper layers when using dynamic SQL rather than stored procedures for object persistence. Many software architects and developers don't consider DataSets to qualify as a DAL because they are data-centric. DataSets are a relational cache and their data model duplicates all or part of the database structure.

*DataSets implement Martin Fowler's Record Set, Active Record, Data Transfer Object, Optimistic Offline Lock, and Unit of Work patterns and applications that use DataSets commonly employ the Table Module application pattern. [Fowler, Patterns of Enterprise Application Architecture, Addison-Wesley, 2003; [www.martinfowler.com](http://www.martinfowler.com)]*

LINQ to SQL classes are an excellent candidate for implementing a DAL. LINQ to SQL's central object is the `DataContext` that manages the connection between the SQL Server 200x or SQL Server Compact Edition 3.5+ database. `DataContext` stores `Table< TEntity >` generic collections where `TEntity` is the singularized name of the corresponding persistence table. For example, the `Customers` table persists `Customer` objects in a `Table< Customer >` collection of type `Customer`. The `Table< TEntity >` type is very versatile; it implements `IEnumerable< T >`, `IQueryable< T >`, `IEnumerable`, `IQueryable`, `IQueryProvider`, `ITable`, and `IListSource` interfaces. Implementing `IListSource` enables databinding to ASP.NET's `LinqDataSource` and Windows forms' `BindingSource`, either of which bind to other databound UI controls.

## The LINQ to SQL Query Pipeline

You execute LINQ query expressions against one or more `Table< TEntity >` objects to generate an `IQueryable< T >` query definition object that an expression tree in the query pipeline translates to parameterized, dynamic T-SQL statements (see Figure 5-4). Translating a chain of SQO lambda expressions (in method call syntax) to an expression tree enables treatment of the query as data rather than code, so the compiler can perform the LINQ to T-SQL translation. The `DataContext` manages the `SqlClient` or `SqlClientCe` connection to the SQL Server or SQLServer CE instance, so you don't need to write connection-related code. In fact, some ASP.NET databinding scenarios (theoretically) require no data-related source code at all. The last step in the query pipeline is executing the query and assembling the rowset(s) returned by the `SqlDataReader` object(s) into `IEnumerable< T >` sequences. If your code changes only parameter values, you can compile the T-SQL query to reduce execution time of successive

## Chapter 5: Using LINQ to SQL and the LinqDataSource

queries. Databinding is the subject of the later “ASP.NET Databinding with the LinqDataSource Control” and “Databinding to Windows Form Controls” sections.

### The LINQ to SQL Query Pipeline

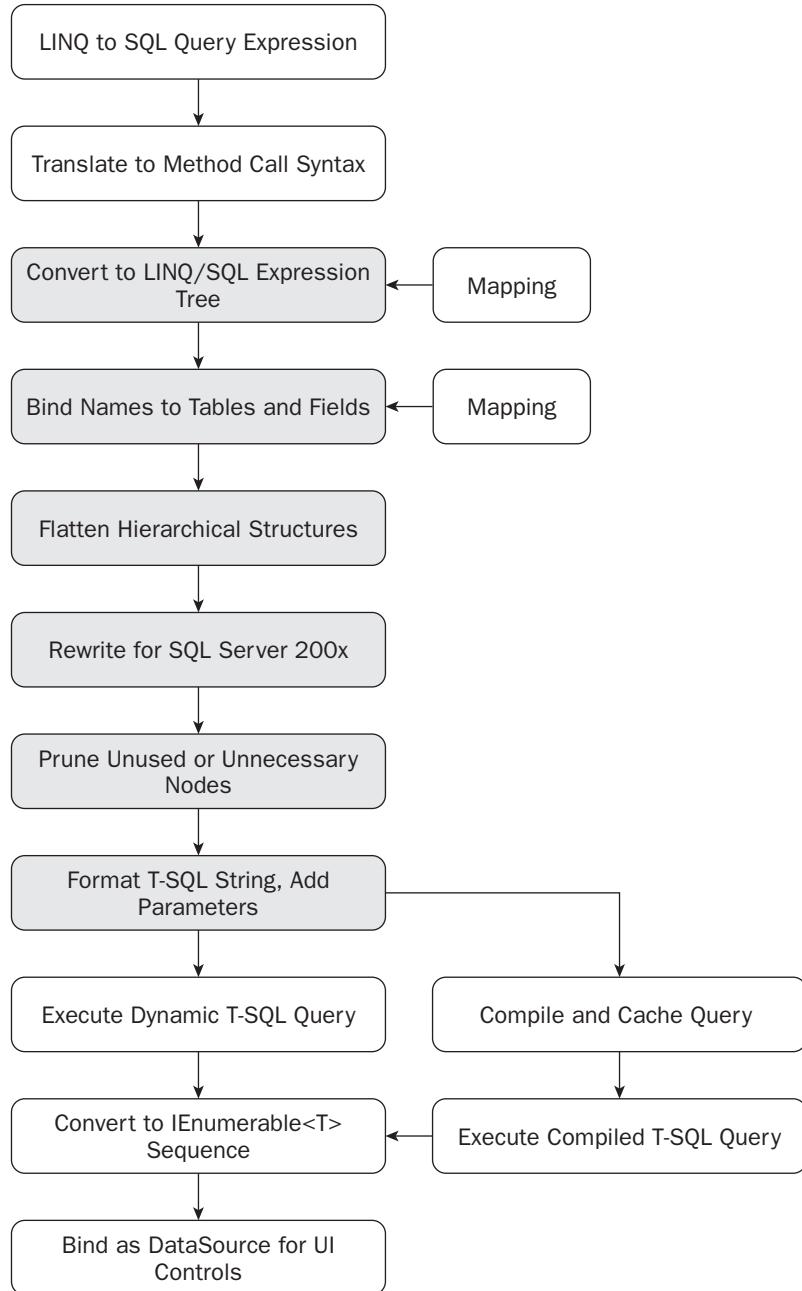


Figure 5-4

## Part III: Applying Domain-Specific LINQ Implementations

---

Following are simple C# 3.0 and VB 9.0 LINQ queries against the Order object that project selected properties of orders shipped to U.S. addresses:

### C# 3.0 Query Expression

```
var query = from o in dcOrders.Orders
            where o.ShipCountry == "USA"
            orderby o.OrderID descending
            select new { o.OrderID, o.CustomerID, EmployeeID,
                        o.ShippedDate, o.ShipCountry
                    };
```

### VB 9.0 Query Expression

```
Dim Query = From o In dcOrders.Orders _
            Where o.ShipCountry = "USA" _
            Order By o.OrderID Descending _
            Select o.OrderID, o.CustomerID, o.EmployeeID, _
                   o.ShippedDate, o.ShipCountry
```

The expression tree for the preceding LINQ query expressions returns the following parameterized T-SQL query with its parameter value:

### T-SQL Query

```
SELECT [t0].[OrderID], [t0].[CustomerID], [t0].[EmployeeID], [t0].[ShippedDate]
FROM [dbo].[Orders] AS [t0]
WHERE [t0].[ShipCountry] = @p0
ORDER BY [t0].[OrderID] DESC
-- @p0: Input String (Size = 3; Prec = 0; Scale = 0) [USA]
```

*The DataContext.Log property, which enables logging the T-SQL statements sent to the server, adds -- @p#0: ... parameter type and value expressions following the query.*

The DataContext's IdentityManager service assigns a unique ID to each business object instance, a process called *uniquing*, which enables the service to ensure that only one copy of any object is in memory at any time. The unique ID usually is a combination of the entity name and primary key value. The DataContext lazy loads associated object instances when needed by multi-table queries only. Lazy loading, which LINQ to SQL calls *Delay Loading*, improves performance and minimizes resource consumption at the expense of periodically opening a myriad of pooled database connections to populate associations. The DataContext's DeferredLoadingEnabled and LoadOptions property values, as well as the Delay Loading setting for Table< TEntity > members determine how and when the DataContext populates EntityRefs and EntitySets.

*Object IDs have DataContext scope. Therefore, caching the DataContext in a form-level or global variable also improves performance of Windows form projects that execute multiple queries against the same set of tables.*

The change tracking service responds to the Table< TEntity >'s PropertyChanged event to record changes to entity property values. There's also a PropertyChanging event that lets you add validation logic for property values in a partial class file. Each entity class implements partial extensibility methods for OnPropertyNameChanging({type value|value As Type}) and OnPropertyNameChanged

events. The `OnValidate` partial method fires on invoking the `DataContext.SubmitChanges()` method but before persisting the entity's changed data to the underlying tables.

In addition to the usual `Add(TEntity)` and `Remove(TEntity)` methods, the `Table< TEntity >` collection has `Attach(TEntity)` and `Detach(TEntity)` methods to support detached modifications. Setting the `DataContext`'s `Serialization Mode` property to `Unidirectional` in the O/R Designer adds `DataContract` and `DataMember` attributes to all `Table< TEntity >` classes and their members to serialize and deserialize objects to XML documents sent in Windows Communication Foundation (WCF) messages for service-based data-tier implementations. WCF substitutes the new `DataContractSerializer` (DCS), which recognizes the `DataContract` and `DataMember` attributes, for the `XmlSerializer` used by .NET 2.0 and earlier Web services.

## ***Adding, Updating, and Removing Objects***

Adding a new instance to a `Table< TEntity >` collection is easy: Create a new instance of `TEntity`, populate the `NewEntity`'s property values and invoke the `DataContext.Table.Add(NewEntity)` method to add it to the collection. Then apply the `DataContext.SubmitChanges()` method to generate and execute the parameterized T-SQL `INSERT` statement that persists the new object. Updating and removing objects requires issuing a query expression with a `Where` clause restriction to obtain references to one or more objects. For the more common, single-instance case, an update simply requires changing property values, while deletion involves invoke the `DataContext.Table.Remove(EntityToDelete)` method; either operation requires invoking the `DataContext.SubmitChanges()` method to persist the changes to the database. The following examples illustrate queries to add, update, and delete `Order_Detail` LINQ to SQL objects:

### **C# 3.0 Methods**

```
public bool InsertDetail(int orderId, int productId, Int16 quantity,
    Decimal unitPrice, Single discount) {
    using (NorthwindDataContext dcNwind = new NorthwindDataContext(conn)) {
        if (dcNwind.Order_Details.Count(c => c.OrderID == orderId
            && c.ProductID == productId) == 0) {
            Order_Detail inserted = new Order_Detail();
            inserted.OrderID = orderId;
            inserted.ProductID = 1;
            inserted.Quantity = 12;
            inserted.UnitPrice = 15.00M;
            inserted.Discount = 0;
            dcNwind.Order_Details.Add(inserted);
            dcNwind.SubmitChanges();
            return true;
        }
        return false;
    }
}

public bool UpdateDetail(int orderId, int productId, Int16 quantity,
    Decimal unitPrice, Single discount) {
    using (NorthwindDataContext dcNwind = new NorthwindDataContext(conn)) {
        Order_Detail updated = dcNwind.Order_Details.
            SingleOrDefault(c => c.OrderID == orderId && c.ProductID == productId);
        if (updated != null) {
            updated.Quantity = quantity;
            updated.UnitPrice = unitPrice;
            updated.Discount = discount;
            dcNwind.SubmitChanges();
            return true;
        }
    }
}
```

## Part III: Applying Domain-Specific LINQ Implementations

---

```
    if (updated != null) {
        updated.OrderID = orderId;
        updated.ProductID = productId;
        updated.Quantity = quantity;
        updated.UnitPrice = unitPrice;
        updated.Discount = discount;
        dcNwind.SubmitChanges();
        return true;
    }
    return false;
}

public bool DeleteDetail(int orderId, int productId) {
    using (NorthwindDataContext dcNwind = new NorthwindDataContext(conn)) {
        Order_Detail deleted = dcNwind.Order_Details.
            SingleOrDefault(d => d.OrderID == orderId && d.ProductID == productId);
        if (deleted != null) {
            dcNwind.Order_Details.Remove(deleted);
            dcNwind.SubmitChanges();
            return true;
        }
        return false;
    }
}
```

### VB 9.0 Methods

```
Public Function InsertDetail(ByVal orderId As Integer, _
                            ByVal productId As Integer, _
                            ByVal quantity As Int16, _
                            ByVal unitPrice As Decimal, _
                            ByVal discount As Single) As Boolean
    Using dcNwind As NorthwindDataContext = New NorthwindDataContext(conn)
        If dcNwind.Order_Details.Count(Function(d) d.OrderID = orderId _ 
            AndAlso d.ProductID = productId) = 0 Then
            Dim inserted As Order_Detail = New Order_Detail()
            inserted.OrderID = orderId
            inserted.ProductID = 1
            inserted.Quantity = 12
            inserted.UnitPrice = 15D
            inserted.Discount = 0
            dcNwind.Order_Details.Add(inserted)
            dcNwind.SubmitChanges()
            Return True
        Else
            Return False
        End If
    End Using
End Function

Public Function UpdateDetail(ByVal orderId As Integer, _
                           ByVal productId As Integer, _
```

## Chapter 5: Using LINQ to SQL and the LinqDataSource

---

```
        ByVal quantity As Int16, _
        ByVal unitPrice As Decimal, _
        ByVal discount As Single) As Boolean
Using dcNwind As NorthwindDataContext = New NorthwindDataContext(conn)
    Dim updated As Order_Detail = _
        dcNwind.Order_Details.SingleOrDefault(Function(d) d.OrderID = _
            orderId AndAlso d.ProductID = productId)
    If updated IsNot Nothing Then
        updated.OrderID = orderId
        updated.ProductID = productId
        updated.Quantity = quantity
        updated.UnitPrice = unitPrice
        updated.Discount = discount
        dcNwind.SubmitChanges()
        Return True
    Else
        Return False
    End If
End Using
End Function

Public Function DeleteDetail(ByVal orderId As Integer, _
                           ByVal productId As Integer) As Boolean
    Using dcNwind As NorthwindDataContext = New NorthwindDataContext(conn)
        Dim deleted As Order_Detail = _
            dcNwind.Order_Details.SingleOrDefault(Function(d) d.OrderID = _
                orderId AndAlso d.ProductID = productId)
        If deleted IsNot Nothing Then
            dcNwind.Order_Details.Remove(deleted)
            dcNwind.SubmitChanges()
            Return True
        Else
            Return False
        End If
    End Using
End Function
```

*Error-handling code has been omitted in the preceding listings for brevity. Source code for the current ASP.NET Web site example is in the \WROX\ADONET\ADONET\CS \LINQtoSQLDataAccessLayer and ...\\VB\\ LINQtoSQLDataAccessLayer folders. LINQ to SQL code is in the NwindSQL.cs and NwindSQL.vb files of the solution's NwindLinqSqlCS and NwindLinqSqlVB class libraries. The projects assume that the Northwind sample database is preattached to a local instance of SQL Server 2005 Express named SQLEXPRESS.*

Each method tests the Order Details table for the existence of the corresponding record before performing its operation on a new or existing object and persisting the changes to the underlying database. Invoking `DataContext.SubmitChanges()` starts a transaction, the `System.Linq.ChangeProcessor` class locates and organizes the objects with pending changes, and passes them to the query pipeline for translation to T-SQL and execution, as shown in Figure 5-5. Alternatively, you can enlist the `DataContext` in an open transaction.

## Part III: Applying Domain-Specific LINQ Implementations

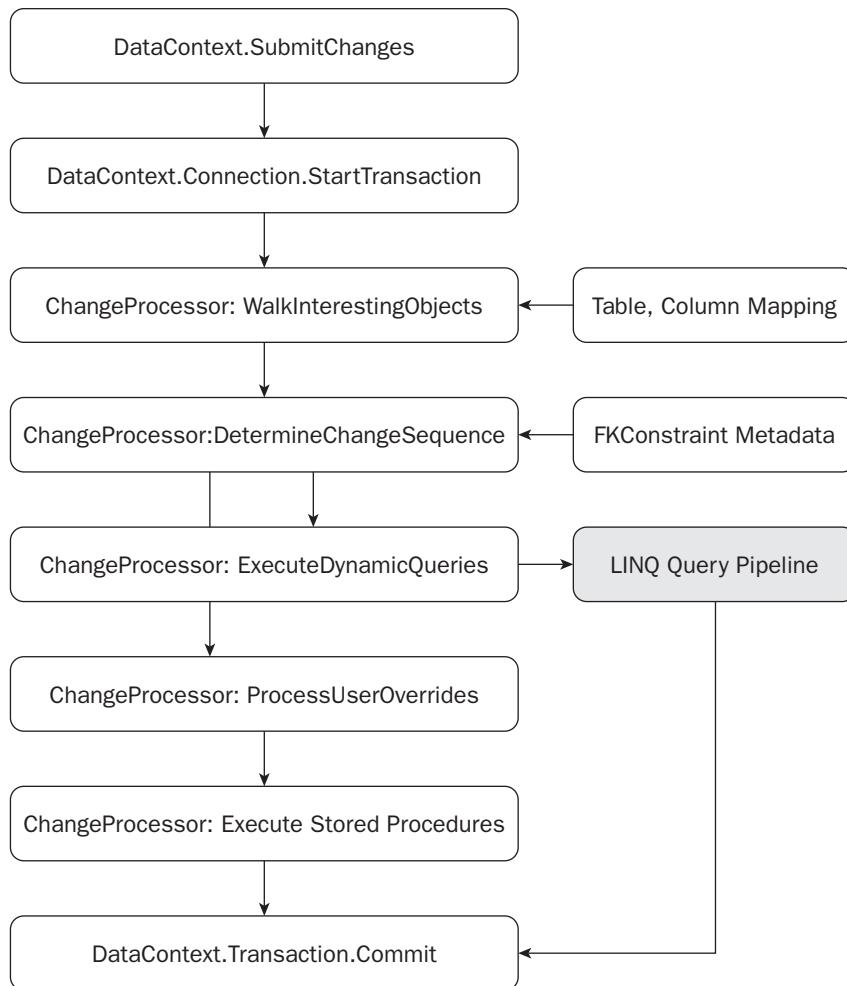


Figure 5-5

Following are the T-SQL `INSERT`, `UPDATE`, and `DELETE` statements with parameter values that the query pipeline translates from the expression trees for the preceding code examples:

### T-SQL Batch Query

```
INSERT INTO [dbo].[Order Details]([OrderID], [ProductID], [UnitPrice], [Quantity], [Discount]) VALUES (@p0, @p1, @p2, @p3, @p4)
-- @p0: Input Int32 (Size = 0; Prec = 0; Scale = 0) [11104]
-- @p1: Input Int32 (Size = 0; Prec = 0; Scale = 0) [1]
-- @p2: Input Currency (Size = 0; Prec = 19; Scale = 4) [15.00]
-- @p3: Input Int16 (Size = 0; Prec = 0; Scale = 0) [12]
-- @p4: Input Single (Size = 0; Prec = 0; Scale = 0) [0]

UPDATE [dbo].[Order Details]
SET [UnitPrice] = @p5
```

## Chapter 5: Using LINQ to SQL and the LinqDataSource

```
WHERE ([OrderID] = @p0) AND ([ProductID] = @p1) AND ([UnitPrice] = @p2) AND  
([Quantity] = @p3) AND ([Discount] = @p4)  
-- @p0: Input Int32 (Size = 0; Prec = 0; Scale = 0) [11106]  
-- @p1: Input Int32 (Size = 0; Prec = 0; Scale = 0) [1]  
-- @p2: Input Currency (Size = 0; Prec = 19; Scale = 4) [15.0000]  
-- @p3: Input Int16 (Size = 0; Prec = 0; Scale = 0) [12]  
-- @p4: Input Single (Size = 0; Prec = 0; Scale = 0) [0]  
-- @p5: Input Decimal (Size = 0; Prec = 29; Scale = 4) [17.50]  
  
DELETE FROM [dbo].[Order Details] WHERE ([OrderID] = @p0) AND ([ProductID] = @p1)  
AND ([UnitPrice] = @p2) AND ([Quantity] = @p3) AND ([Discount] = @p4)  
-- @p0: Input Int32 (Size = 0; Prec = 0; Scale = 0) [11104]  
-- @p1: Input Int32 (Size = 0; Prec = 0; Scale = 0) [1]  
-- @p2: Input Currency (Size = 0; Prec = 19; Scale = 4) [15.0000]  
-- @p3: Input Int16 (Size = 0; Prec = 0; Scale = 0) [12]  
-- @p4: Input Single (Size = 0; Prec = 0; Scale = 0) [0]
```

UnitPrice is the only column updated and is highlighted with its parameter value in the preceding listing generated by the `DataContext.Log` property. To log all T-SQL batches generated by LINQ queries and ChangeProcessor activity, assign the `DataContext.Log` property to a `StreamWriter` that appends the batches to a file with the following code:

### C# 3.0

```
StreamWriter swLog = new StreamWriter("Log.sql", true);  
DataContext.Log = swLog;  
// ... Code to generate T-SQL batches  
[DataContext.SubmitChanges();]  
swLog.Flush();  
swLog.Close();
```

### VB 9.0

```
Dim swLog As StreamWriter = New StreamWriter("Log.sql", True)  
dcNwind.Log = swLog  
'... Code to generate T-SQL batches  
dcNwind.SubmitChanges()  
[DataContext.SubmitChanges()]  
swLog.Flush()  
swLog.Close()
```

ASP.NET's default location for the log file is \Program Files\Visual Studio 9.0\Common7\IDE.

## Detecting and Resolving Concurrency Conflicts

LINQ to SQL supports optimistic concurrency for object updates and deletions. The default conflict detection method is comparison of original property values stored by the `DataContext` with current database values. The `@p0` through `@p4` parameters in the preceding query examples supply the original values for the preceding `UPDATE` and `DELETE` statements. If the values match those for the table row, the update or deletion proceeds; if not, the operation throws a concurrency exception. You can specify how frequently members are tested for concurrency conflicts by setting the `Update Check` attribute value to `Always`, `WhenChanged`, or `Never` in the O/RM designer's properties sheet for the member. The default is `Always` for data types other than `image`, `text`, `ntext`, `binary`, and `varbinary`.

## Part III: Applying Domain-Specific LINQ Implementations

If your source tables have a `timestamp` field, you can set the `Time Stamp` attribute value to `True` in the member's property sheet to substitute `timestamp` tests for property value comparisons and gain improved performance (refer to Figure 5-3 right). Adding a `timestamp` field enables persisting object modifications without original values because updates or deletions succeed only if the `timestamp` value for the object and row are the same. You lose the benefit of replacing data in changed fields only, but sending original values for most or all fields usually outweighs the advantage of value-based concurrency management. If the database's schema is under your control or its DBA is amenable, add a `timestamp` field to each table.

You can update or delete multiple objects in an iterator loop and then call the `DataContext.SubmitChanges()` method. `SubmitChanges()` uses the `DataContext`'s change tracking feature to batch the T-SQL `UPDATE` and `DELETE` operations to the server. Alternatively, you can invoke the `Table.AddAll(IEnumerable< TEntity >)` or `Table.RemoveAll(IEnumerable< TEntity >)` method to add or remove multiple instances with a LINQ query result as the method's argument. The `DataContext.SubmitChanges(ConflictMode)` overload lets you choose between `FailOnFirstConflict`, which terminates the batch when the first concurrency conflict occurs, or `ContinueOnConflict`, which attempts to complete all modifications, removals, or both and reports conflicts at the end of the batch. The `DataContext.MemberConflicts` collection returns a `ObjectChangeConflict` collection with a `MemberChangeConflict` item for each conflicting entity and member.

Figure 5-6 shows a Windows form LINQ to SQL client displaying a message box that lets data entry operators choose one of the three values of the `RefreshMode` enum — `KeepChanges`, `KeepCurrentValues`, or `OverwriteCurrentValues` — to resolve the concurrency conflict.

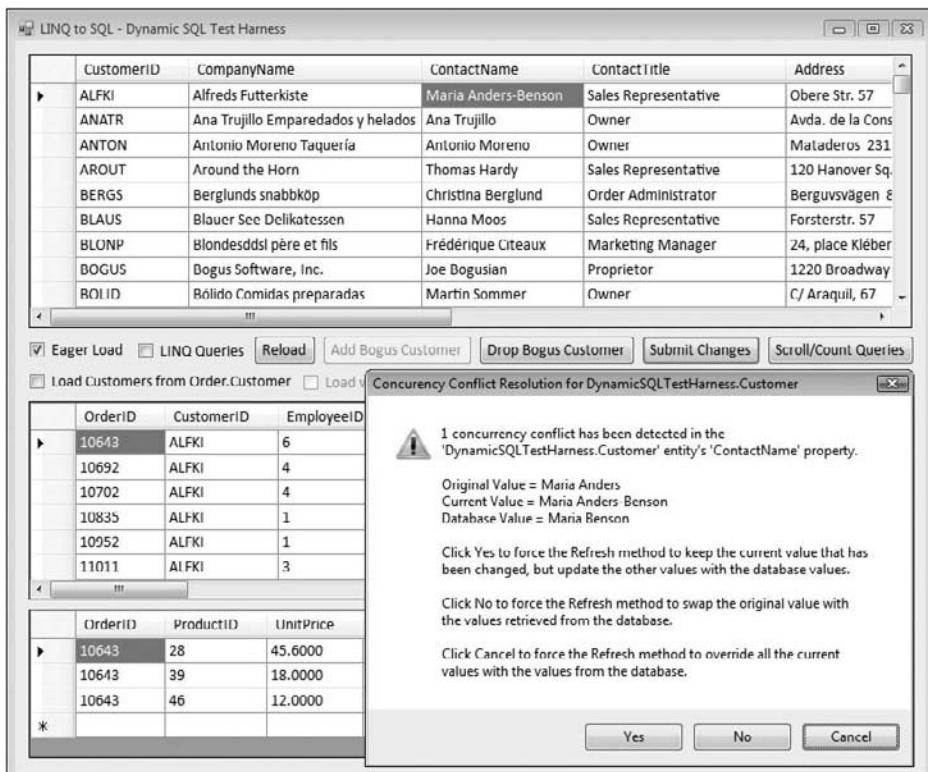


Figure 5-6

## Chapter 5: Using LINQ to SQL and the LinqDataSource

---

The exception handler for the `SubmitChanges()` method calls the following function when the method encounters a `ChangeConflictException`:

### C# 3.0

```
private void ProcessConcurrencyConflicts()
{
    // Enumerate and handle the change conflicts for updates and deletions
    foreach (ObjectChangeConflict ccObject in dcNwind.ChangeConflicts)
    {
        try
        {
            foreach (MemberChangeConflict ccMember in ccObject.MemberConflicts)
            {
                int conflictCount = dcNwind.ChangeConflicts.Count;
                string message = null;
                if (conflictCount == 1)
                {
                    message = "1 concurrency conflict has ";
                }
                else
                {
                    message = conflictCount.ToString() + " concurrency conflicts have ";
                }
                message += "been detected in the '" + ccObject.Object.ToString() +
                           "' entity's '" + ccMember.Member.Name + "' property.\r\n\r\nOriginal
                           Value = " + ccMember.OriginalValue.ToString() + "\r\nCurrent Value = "
                           + ccMember.CurrentValue.ToString() + "\r\nDatabase Value = " +
                           ccMember.DatabaseValue.ToString() + "\r\n\r\nClick Yes to force the " +
                           "Refresh method to keep the current value that has been changed, " +
                           "but update the other values with the database values\r\n\r\n" +
                           "Click No to force the Refresh method to swap the original value " +
                           "with the values retrieved from the database.\r\n\r\n" +
                           "click Cancel to force the Refresh method to override all the " +
                           "current values with the values from the database.";

                // Resolve each member conflict
                int resolveAction = MessageBox.Show(message,
                           "Concurrency Conflict Resolution for " + ccObject.Object.ToString(),
                           MessageBoxButtons.YesNoCancel, MessageBoxIcon.Exclamation);
                if (resolveAction == (int)DialogResult.Yes)
                {
                    ccMember.Resolve(RefreshMode.KeepChanges);
                }
                else if (resolveAction == (int)DialogResult.No)
                {
                    ccMember.Resolve(RefreshMode.KeepCurrentValues);
                }
                else
                {
                    ccMember.Resolve(RefreshMode.OverwriteCurrentValues);
                }
            }
        }
    }
}
```

## Part III: Applying Domain-Specific LINQ Implementations

---

```
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, strTitle, MessageBoxButtons.OK,
                           MessageBoxIcon.Exclamation);
        }
    }
}
```

Although it isn't likely, there can be multiple entities of various types in a single persistence operation, each with multiple properties in conflict. Thus, your code must iterate two nested loops to continue execution with all conflicts resolved.

*The preceding function is part of the C# version of the DynamicSQLTestHarness.sln project in the \Wrox\ADONET\Chapter05\CS\DynamicSQLTestHarness folder. The VB listing is omitted for brevity.*

Adding or deleting an object that has *m:1* associations with other objects, *1:n* associations with collections, or both, requires updating the associations' *EntyRefs* and *EntitySets*. For example, adding or deleting a Northwind *Order* object requires updating *Order.Customer*, *Order.Employee*, and *Order.Shipper* *EntityRefs*, as well as *Customer.Orders*, *Employee.Orders*, and *Shipper.Orders* *EntitySets*. LINQ to SQL handles all required association and primary/foreign-key fixups for *Table< TEntity >* entity sets automatically, including adding object IDs, such as *OrderID*, generated for *INSERT* operations by SQL Server *int identity* columns. However, LINQ to SQL doesn't cascade updates or deletions to maintain referential integrity automatically. For example, you must delete all related *Order\_Detail* and then *Order* objects before deleting a *Customer* object, unless you've enabled cascading deletions for the Northwind database's foreign-key constraints.

## **Substituting Stored Procedures for Dynamic SQL**

There is substantial controversy among developers and database administrators (DBAs) regarding the desirability or necessity of limiting applications' database access to stored procedures. The arguments in favor of stored procedures usually are better access control because users must be granted explicit execute permissions on each stored procedure, heightened security by precluding "SQL injection attacks," improved performance because "stored procedures are compiled" and isolation of applications from database schema changes. Following are common replies to these four contentions:

- 1. Access control.** It's true that members of the database's default *db\_datareader* and *db\_datawriter* roles aren't granted the execute privilege for stored procedures, but few DBAs add user accounts to enable these groups, which have database-wide permissions. The more secure approach is to create a custom database or application role, grant the role permissions to execute related stored procedures, and then add accounts for users, processes, or both to the role.
- 2. SQL injection attacks.** These attacks depend on the ability of an attacker to substitute executable T-SQL statements for a literal variable in a dynamic T-SQL statement. The accepted remedy for this vulnerability is to use T-SQL parameters for all variables. LINQ to SQL uses parameterized queries for all dynamic T-SQL statement sent to the server.
- 3. Performance.** SQL Server versions 6.5 and earlier partially compiled query execution plans for stored procedures when created and used the compiled plan to speed execution; version 7.0 and later no longer compile stored procedures. Instead SQL Server 7.0 and later compile execution plans for all stored procedures and SQL statements and retain the compiled plans in the procedure cache. There has been no difference between the execution speed of a stored

procedure and its dynamic T-SQL parameterized prepared statement counterpart since the release of SQL Server 7.0.

4. **Schema change isolation.** With stored procedures, a database schema change requires all stored procedures that access the changed tables to be rewritten and tested. If the change adds functionality, a separate set of stored procedures must be written and tested to support applications that need the new features while retaining compatibility with deprecated versions. A DAL with code-generation capability, such as LINQ to SQL, can make schema change adaptation semi-automatic. Substituting the DAL for stored procedures concentrates reaction to schema changes in single location, the DAL's assembly.

If you *must* use stored procedures rather than the default dynamic SQL batches, the following two sections explain how to do this.

*Stored procedure code is in the NwindSProc.cs and NwindSProc.vb files of the solution's NwindLinqSProcsCS and NwindLinqSProcsVB class libraries. The projects require executing the CreateDataLayerSProcs.sql script from the \WROX\ADONET\Chapter05 folder in SQL Server Management Studio [Express] for the Northwind database to display pages with SP in their name that use the stored procedure class libraries. A DropDataLayerSProcs.sql script is provided to remove them.*

### Using a **SELECT** Stored Procedure to Return an **ISingleResult< TEntity >**

To substitute a stored procedure for a dynamic T-SQL batch to hydrate objects from the database table you selected to persist an entity, drag the stored procedure node from the Server Explorer's Stored Procedures group and drop it on the appropriate entity class widget. Alternatively, drag the stored procedure directly to the Method pane at the right of the designer's main pane (refer to Figure 5-2), open the properties sheet for the stored procedure, and change the Return Type property to Customer.

*Creating an entity class widget from scratch, rather than from an existing database table, requires dragging a Class widget from the Toolbox, adding properties and setting their data type. After you've added related classes, add Association widgets and set their property values.*

A SELECT stored procedure created in the Northwind database named `usp_GetCustomers` and dropped on the Customer widget adds the following method to the autogenerated `NorthwindDataContext` partial class:

#### C# 3.0

```
[Function(Name = "dbo.usp_GetCustomers")]
public ISingleResult<Customer> usp_GetCustomers() {
    IExecuteResult result =
        this.ExecuteMethodCall(this, ((MethodInfo)(MethodInfo.GetCurrentMethod())));
    return ((ISingleResult<Customer>)(result.ReturnValue));
}
```

#### VB 9.0

```
<FunctionAttribute(Name:="dbo.usp_GetCustomers")> _
Public Function usp_GetCustomers() As ISingleResult(Of Customer)
    Dim result As IExecuteResult =
        Me.ExecuteMethodCall(Me, CType(MethodInfo.GetCurrentMethod(), MethodInfo))
    Return CType(result.ReturnValue, ISingleResult(Of Customer))
End Function
```

## Part III: Applying Domain-Specific LINQ Implementations

`ISingleResult<T>` implements `IEnumerable<T>`, so you can add a function in a partial class file to invoke the `ToList()` method to create a generic `List<T>` collection that you can bind to an `ObjectDataSource` control or directly to a data-aware Web control. `ISingleResult<T>` is a *much* simpler and less versatile object than `Table< TEntity >`.

### Using **INSERT**, **UPDATE**, and **DELETE** Stored Procedures

To substitute stored procedures for default T-SQL batches, drag the appropriate stored procedure nodes from Server Explorer's Stored Procedures group to the Method pane. Right-click an entity widget and choose Manage Behavior to open the eponymous dialog. Open the Class list to choose the entity to update; select Insert, Update, or Delete in the Behavior list; select the Customize option button; and choose the corresponding stored procedure from the Customize list (see Figure 5-7). If your method argument names are similar to the entity's property names, the dialog matches them automatically. Click OK to save the changes in the mapping file. Otherwise, use the Class Properties lists to select the appropriate property for the selected method argument. You can return to the use of dynamic SQL batches for updates by selecting the Use Runtime option button.

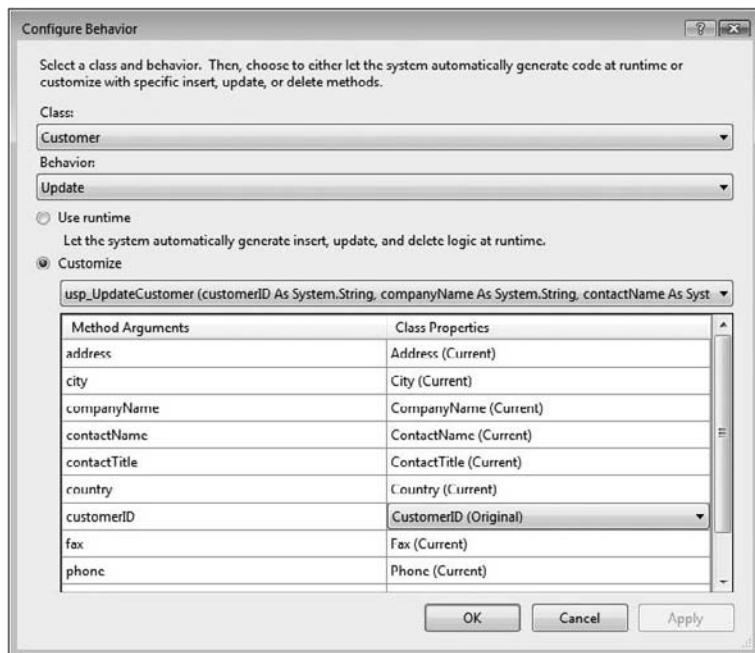


Figure 5-7

*LINQ to SQL DALs adopt Fowler's DataMapper application pattern and implement the Unit of Work pattern by change tracking, Metadata Mapping by DataContext and Table< TEntity >s, Identity Map by unquing, Foreign Key Mapping by EntityRef and EntitySet, and Lazy Loading by Delay Loading. The current lack of support for Detached change tracking prevents LINQ to SQL DALs from implementing Service Layer and Data Transfer Objects patterns, but the Unit of Work pattern doesn't require support for remoting.*

### **Moving the LINQ to SQL Layer to a Middle Tier**

Dinesh Kulkarni, LINQ to SQL's program manager during VS 2008's beta stages, wrote in an October 15, 2007 blog post entitled *LINQ to SQL: What's NOT in RTM (V1)*: "Out-of-the-box multi-tier story. Yes, we do have good Attach( ) APIs but we don't yet have simple roundtripping and change tracking on the client." (See <http://blogs.msdn.com/dinesh.kulkarni/archive/2007/10/15/linq-to-sql-what-is-not-in-rtm-v1.aspx>). The upshot of this admission is that it's difficult — but not impossible — to use LINQ to SQL as a data-access tier in a project that implements service-oriented architecture.

LINQ to SQL v.1 in VS 2008 doesn't support in detached mode the type of change tracking that's required for simple physical separation of a DAL into an independent data access tier. DataSets handle change tracking by maintaining an original version of each DataRow and adding a modified version that reflects data changes, so change tracking is embedded in the data. Thus the serialized DataSet objects that travel between process boundaries track their own changes. Using this approach with LINQ to SQL objects would violate persistence ignorance rules.

The technique that the LINQ to SQL team had proposed to enable detached change tracking to LINQ to SQL was a serializable "mini-connectionless DataContext" to hold changes on the client side. The mini-DataContext would be sent to the client with original object graph, returned by the client with the changes, and finally attached to a new, full-size DataContext object to persist the changes. Unfortunately, the mini-DataContext won't arrive in VS 2008. Until the SQL team implements the mini-DataContext or its equivalent, or provides sample code to emulate the DataSet's detached (fully disconnected) capabilities, enabling disconnected mode with LINQ to SQL in a separate tier will remain a major hurdle for .NET developers.

*Another hurdle for the use of LINQ to SQL in n-tier scenarios is LINQ to SQL v1's limitation of unidirectional-only serialization, which prevents serializing object graphs with EntitySets and EntityRefs with theDataContractSerializer. A change to the DataContractSerializer in VS 2008 SP1 enables bidirectional serialization but neither the O/R Designer or SqlMetal.exe apply the DataMember attribute to ResultSet members. You can overcome this limitation by applying the technique described in the "Bidirectional Serialization of LINQ to SQL Object Graphs with Damien Guard's T4 Template in VS 2008 SP1" blog post (<http://oakleafblog.blogspot.com/2008/07/bidirectional-serialization-of-linq-to.html>).*

# ASP.NET Databinding with the LinqDataSource Control

Simple drag-and-drop databinding of UI controls of Windows and Web forms has been a feature of VS since version 7.0 (2002). LINQ to SQL supports databinding of its `DataContext` or entity sets to the following types of data-aware ASP.NET Web controls:

- ❑ **LinqDataSource** is a new Web data source control that binds LINQ to SQL `DataContext` `.Table< TEntity >` objects only. This control combines the design-time simplicity of the client/server-style `SqlDataSource` control with the three-layer architecture of the `ObjectDataSource` control. The `LinqDataSource` enables codeless databinding of all data-aware ASP.NET UI controls; it processes sorting with server-side `OrderBy` and paging operations. (Server-side paging requires SQL Server 2005 or later because it uses the `ROW_NUMBER` function.) Updates, deletions and selection with `GridView` controls, as well as updates, insertions, and deletions with `DetailsView` or templated `FormView` controls, require no enabling code. The presentation (UI) layer has knowledge of the underlying database structure and connection string because the `DataContext` in the DAL's assembly must have `public` scope to provide the `LinqDataSource` control's `DataSource`.
- ❑ **ObjectDataSource** is a traditional Web control whose data source can be a `Table< EntityName >`, LINQ `IEnumerable< T >` sequence, or other collection that implements `IEnumerable< T >`, such as `List< T >`. When using a LINQ to SQL DAL, it's not necessary for the `ObjectDataSource` control to gain access to the `DataContext`; `ObjectDataSource` controls connect directly to the `Table< EntityName >` entity sets, such as `NorthwindDataContext.Customers`, so you can change the `DataContext`'s `Access` property from `Public` to `Internal`. The `ObjectDataSource` reduces the amount of custom code required to use business object methods or stored procedures to update objects, compared with direct databinding of UI controls, and plugs the perceived leak in the object abstraction that direct access to the `DataContext` creates.
- ❑ **GridView**, **DataList**, **ListView**, **DetailView**, and other data-aware UI controls can bind directly to a `List< EntityName >` that you cast from a LINQ `IEnumerable< T >` sequence by invoking the `ToList()` method. You must instantiate a new `DataContext` with code when you don't use a data source control as an intermediary. Set the `ControlId.DataSource()` property value to `List< EntityName >` and invoke the `ControlId.DataBind()` method, which executes the LINQ query immediately. Direct binding is most often used for read-only data access because you need to write a substantial amount of custom code to update data items.

The process of binding `DataContext` classes as the `DataSource` property of the `ObjectDataSource` and data-aware UI controls is the same as that for binding business objects that implement the `DataRow` and `DataRowView` types, so only the `LinqDataSource` is covered in this chapter. Basic familiarity with configuring the `ObjectDataSource`, `GridView`, and other databound controls is assumed.

## **Adding a LinqDataSource to a Page**

The `LinqDataSource` control implements the ASP.NET 2.0 data source control model and resembles VS 2005's `ObjectDataSource` control. Unlike the `ObjectDataSource`, which supports generic business objects that implement custom select, insert, update, and delete methods, the `LinqDataSource` relies on the `DataContext` object to populate and manage object collections, including insert, update, and delete method invocation.

## Chapter 5: Using LINQ to SQL and the LinqDataSource

Binding the LinqDataSource control to a LINQ to SQL DAL's `Table< TEntity >` involves these steps:

1. Add a reference to the assembly containing the LINQ to SQL `DataContext` and classes.
2. Add a LinqDataSource control to the page in Design mode.
3. Right-click the LinqDataSource control, and choose **Configure Data Source** to start the **Configure Data Source Wizard**.
4. In the first wizard dialog, set the control's `DataSource` property to the appropriate `DataContext`, `Northwind.NorthwindDataContext` for this example, and click next to open the **Configure Data Selection** dialog.
5. Select the entity collection name from Tables list, `Orders` for this example, and accept the default \* (all fields) check box selection (see Figure 5-8).

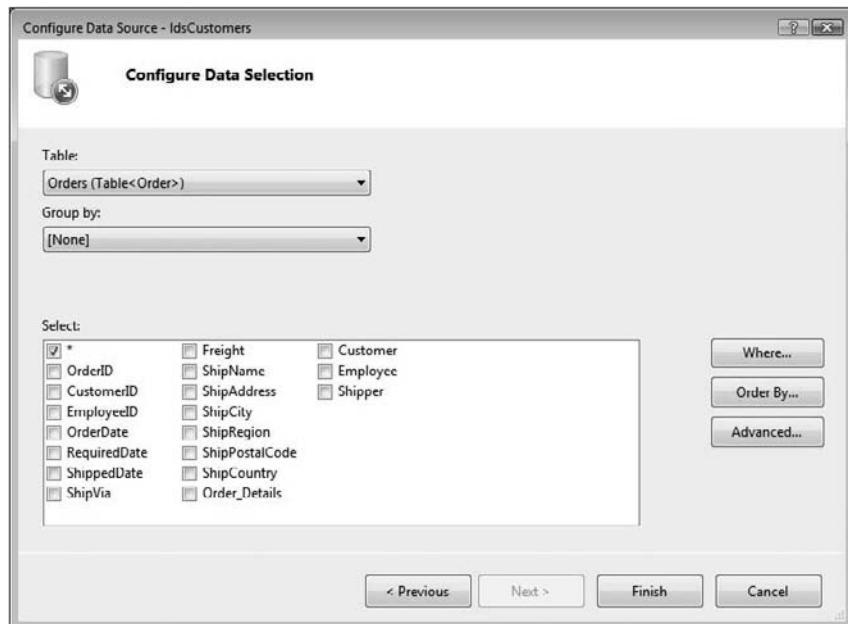


Figure 5-8

*EntityRefs (Customer, Employee, and Shipper for the Order class) and EntitySets (Order\_Details) are included as properties of the selected "Table," which would better have been described in the Configure Data Selection dialog as an Entity Class.*

6. If you want to add a `Where` clause to the LINQ query, click the `Where` button to open the **Configure Where Expression** dialog, define the Column, Operator, (parameter) Source, and Value, if the argument is a literal (see Figure 5-9).

## Part III: Applying Domain-Specific LINQ Implementations

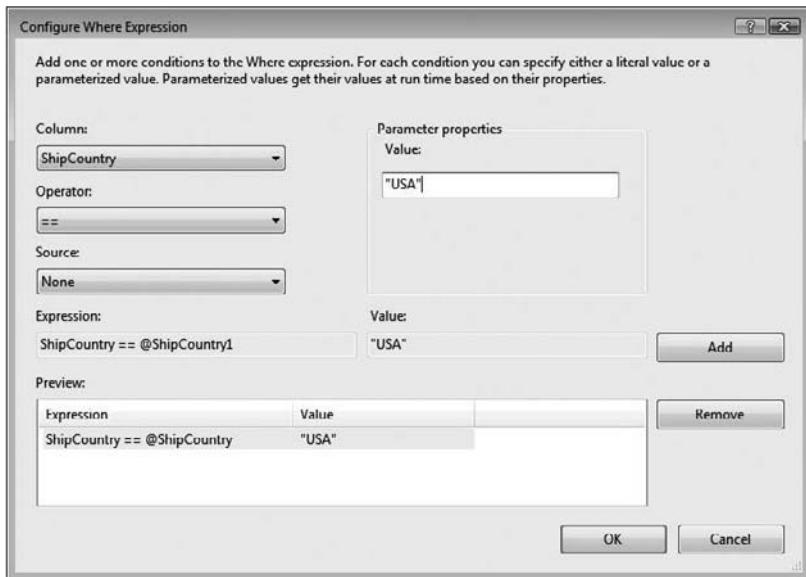


Figure 5-9

7. If you want to apply a non-default sort order to the `Table< TEntity >` collection, click the Order By button, and specify the field(s) and sort order.
8. If you want to execute inserts, updates, and deletions with autogenerated dynamic T-SQL statements, click the Advanced button to open the Advanced Options dialog, mark the appropriate check boxes, and click OK.

*If you didn't accept the default \* check box section in step 5, the Advanced button is disabled.*

9. Click Finish to dismiss the wizard.

Now you can bind a `GridView` or other data-aware Web control to the `LinqDataSource`. Be sure to enable paging of UI controls that are capable of displaying multiple rows. Server-side paging downloads only the number of rows that you specify as the value of the UI control's `PageSize` property.

## **Substituting EntityRef for ForeignKey Values in Databound Web Controls**

Foreign-key values from surrogate primary keys, such as autogenerated integers from `int identity` fields, and derived codes from natural primary keys don't convey much meaning to ordinary users of your Web applications. It's a much better approach to substitute a meaningful name or description for cryptic foreign-key values. Property values of associated entities that have an *m:1* association with the entity can provide the required name or description to most databound Web controls that support the Template Fields feature. An advantage of working with associated entities is that you don't need to join the underlying persistence tables to translate the foreign-key value.

## Chapter 5: Using LINQ to SQL and the LinqDataSource

Substituting Customer, Employee, and Shipper names for the default string and integer foreign-key values requires choosing Edit Fields from the UI control bound to the ObjectDataSource, a GridView control for this example, to open the Fields dialog. Add each EntityRef field from the Available Fields to the Selected Fields list, and click the Convert This Field into a Template Field link button to create the starting block of source code. Optionally delete the bound field for the corresponding foreign-key values (see Figure 5-10).

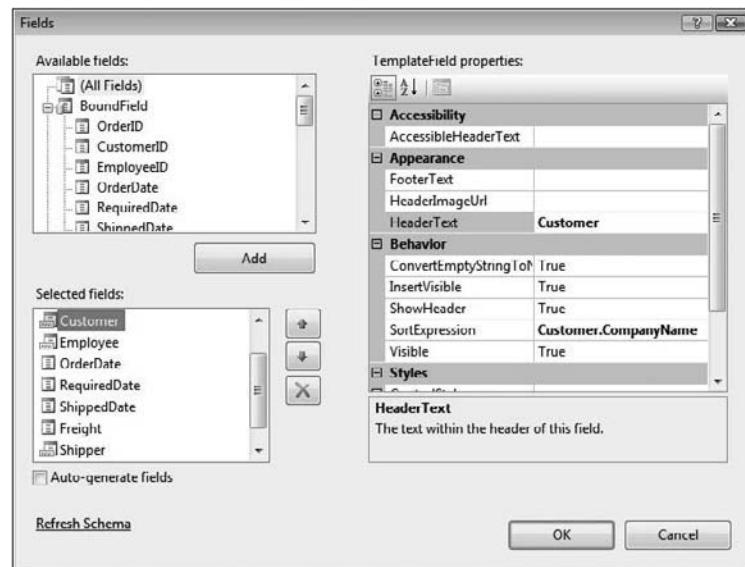


Figure 5-10

You must edit the XHTML code for the template fields to specify the property name of the item to display as the `Text` and `SortExpression` attribute values, as emphasized in the following example for the source code of the first two template fields:

### XHTML 1.0 Transitional

```
<asp:templatefield HeaderText="Customer" SortExpression="Customer.CompanyName">
    <edititemtemplate>
        <asp:TextBox ID="TextBox1" runat="server"
            Text='<%# Eval("Customer.CompanyName") %>'>
        </asp:TextBox>
    </edititemtemplate>
    <itemtemplate>
        <asp:Label ID="Label1" runat="server"
            Text='<%# Eval("Customer.CompanyName") %>'>
        </asp:Label>
    </itemtemplate>
</asp:templatefield>
<asp:templatefield HeaderText="Employee" SortExpression="Employee.LastName">
    <edititemtemplate>
```

## Part III: Applying Domain-Specific LINQ Implementations

```
<asp:TextBox ID="TextBox2" runat="server"
    Text='<%# Eval("Employee.LastName") %>'>
</asp:TextBox>
</edititemtemplate>
<itemtemplate>
    <asp:Label ID="Label2" runat="server"
        Text='<%# Eval("Employee.LastName") %>'>
    </asp:Label>
</itemtemplate>
</asp:templatefield>
```

Eval is emphasized in the preceding code because you replace the Bind verb inserted by code behind the dialog with Eval. Figure 5-11 shows the result of adding the three template fields.

The screenshot shows a Microsoft Internet Explorer window with the title "A Paged Orders List Created from a GridView Bound to a LinqDataSource - Windows Internet Explorer". The address bar shows the URL "http://localhost:50033/OrdersDataSourceVB/Orders.aspx". The page displays a GridView containing 10 rows of order data, each with edit and select links. The columns are: OrderID, Customer, Employee, OrderDate, RequiredDate, ShippedDate, Freight, Shipper, and ShipName. The last row shows a navigation bar with links 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, and an ellipsis (...). At the bottom right of the browser window, there are status bars for "Local intranet | Protected Mode: Off" and "100%".

	OrderID	Customer	Employee	OrderDate	RequiredDate	ShippedDate	Freight	Shipper	ShipName
Edit Select	10263	Ernst Handel	Dodsworth	7/23/1996	8/20/1996	7/31/1996	\$146.06	Federal Shipping	Ernst Handel
Edit Select	10264	Folk och få HB	Suyama	7/24/1996	8/21/1996	8/23/1996	\$3.67	Federal Shipping	Folk och få HB
Edit Select	10265	Blondesdsl père et fils	Fuller	7/25/1996	8/22/1996	8/12/1996	\$55.28	Speedy Express	Blondel père et fils
Edit Select	10266	Wartian Herkku	Leverling	7/26/1996	9/6/1996	7/31/1996	\$25.73	Federal Shipping	Wartian Herkku
Edit Select	10267	Frankenversand	Peacock	7/29/1996	8/26/1996	8/6/1996	\$208.58	Speedy Express	Frankenversand
Edit Select	10268	GROSELLA-Restaurante	Callahan	7/30/1996	8/27/1996	8/2/1996	\$66.29	Federal Shipping	GROSELLA-Restaurante
Edit Select	10269	White Clover Markets	Buchanan	7/31/1996	8/14/1996	8/9/1996	\$4.56	Speedy Express	White Clover Markets
Edit Select	10270	Wartian Herkku	Davolio	8/1/1996	8/29/1996	8/2/1996	\$136.54	Speedy Express	Wartian Herkku
Edit Select	10271	Split Rail Beer & Ale	Suyama	8/1/1996	8/29/1996	8/30/1996	\$4.54	United Package	Split Rail Beer & Ale
Edit Select	10272	Rattlesnake Canyon Grocery	Suyama	8/2/1996	8/30/1996	8/6/1996	\$98.03	United Package	Rattlesnake Canyon Grocery
Edit Select	10273	QUICK-Stop	Leverling	8/5/1996	9/2/1996	8/12/1996	\$76.07	Federal Shipping	QUICK-Stop
Edit Select	10274	Vins et alcools Chevalier	Suyama	8/6/1996	9/3/1996	8/16/1996	\$6.01	Speedy Express	Vins et alcools Chevalier
Edit Select	10275	Magazzini Alimentari Riuniti	Davolio	8/7/1996	9/4/1996	8/9/1996	\$26.93	Speedy Express	Magazzini Alimentari Riuniti
Edit Select	10276	Tortuga Restaurante	Callahan	8/8/1996	8/22/1996	8/14/1996	\$13.84	Federal Shipping	Tortuga Restaurante
Edit Select	10277	Morgenstern Gesundkost	Fuller	8/9/1996	9/6/1996	8/13/1996	\$125.77	Federal Shipping	Morgenstern Gesundkost

Figure 5-11

The *OrdersLinqDataSourceCS.sln* and *OrdersLinqDataSourceVB.sln* sample Web Site projects that demonstrate the code of this and the succeeding section are in the \Wrox\ADONET\Chapter05\CS\OrdersLinqDataSource and ...\VB\OrdersLinqDataSource folders. The projects assume that the Northwind sample database is preattached to a local instance of SQL Server 2005 Express named SQLEXPRESS.

## Eager-Loading EntityRef Values to Reduce Database Server Traffic

Lazy loading minimizes initial memory resources consumed by entity sets at the expense of a roundtrip to the database for *each* EntityRef value. A GridView with a PageSize of 15 Order instances executes the following primary query for the first page:

### T-SQL Query

```
SELECT TOP 15 [t0].[OrderID], [t0].[CustomerID], [t0].[EmployeeID],  
[t0].[OrderDate], [t0].[RequiredDate], [t0].[ShippedDate], [t0].[ShipVia],  
[t0].[Freight], [t0].[ShipName], [t0].[ShipAddress], [t0].[ShipCity],  
[t0].[ShipRegion], [t0].[ShipPostalCode], [t0].[ShipCountry]  
FROM [dbo].[Orders] AS [t0]
```

Then 14 executions of this prepared statement retrieve the associated Customer entities:

### T-SQL Prepared Statement

```
exec sp_executesql N'SELECT [t0].[CustomerID], [t0].[CompanyName],  
[t0].[ContactName], [t0].[ContactTitle], [t0].[Address], [t0].[City],  
[t0].[Region], [t0].[PostalCode], [t0].[Country], [t0].[Phone], [t0].[Fax]  
FROM [dbo].[Customers] AS [t0]  
WHERE [t0].[CustomerID] = @p0',N'@p0 nvarchar(5)',@p0=N'VINET'
```

*The operation hydrates 14 not 15 Customer instances because Hanari Carnes is duplicated in the first 15 orders.*

Finally, seven instances of a similar prepared statement retrieve non-duplicate associated Employee objects and three instances retrieve Shipper objects. It takes 25 T-SQL batches to fill the GridView's 15 rows, which is probably more roundtrips to the server than you'd like the page to make on loading.

You can reduce roundtrips to one by eager-loading the associated EntityRef instances. Eager loading requires populating the `DataContext.LoadOptions` property with a collection of `DataLoadOptions.LoadWith<T>(LambdaExpression)` methods before executing the initial query. Here's the code to cause the equivalent of all 25 preceding batches to execute with a single T-SQL query:

### C# 3.0

```
public partial class NorthwindDataContext {  
    partial void OnCreated() {  
        DataLoadOptions dlOrder = new DataLoadOptions();  
        dlOrder.LoadWith<Order>(o => o.Customer);  
        dlOrder.LoadWith<Order>(o => o.Employee);  
        dlOrder.LoadWith<Order>(o => o.Shipper);  
        this.LoadOptions = dlOrder;  
    }  
}
```

The `OnCreated()` method implements the `partial void OnCreated()` partial method from the Extensibility Method Definitions region of the `ModelName.Designer.cs` file.

## Part III: Applying Domain-Specific LINQ Implementations

---

### VB 9.0

```
Partial Public Class NorthwindDataContext
    Private Sub OnCreated()
        Dim dlOptions As New DataLoadOptions()
        dlOptions.LoadWith(Of Order)(Function(o) o.Customer)
        dlOptions.LoadWith(Of Order)(Function(o) o.Employee)
        dlOptions.LoadWith(Of Order)(Function(o) o.Shipper)
        Me.LoadOptions = dlOptions
    End Sub
End Class
```

Private Sub OnCreated() implements the Partial Private Sub OnCreated() partial method from the ModelName.Designer.vb file.

Here's the single T-SQL batch that returns the first 15 orders and their associated EntityRef objects:

### T-SQL Query

```
SELECT TOP 15 [t0].[OrderID], [t0].[CustomerID], [t0].[EmployeeID],
       [t0].[OrderDate], [t0].[RequiredDate], [t0].[ShippedDate], [t0].[ShipVia],
       [t0].[Freight], [t0].[ShipName], [t0].[ShipAddress], [t0].[ShipCity],
       [t0].[ShipRegion], [t0].[ShipPostalCode], [t0].[ShipCountry], [t2].[test],
       [t2].[CustomerID] AS [CustomerID2], [t2].[CompanyName], [t2].[ContactName],
       [t2].[ContactTitle], [t2].[Address], [t2].[City], [t2].[Region],
       [t2].[PostalCode], [t2].[Country], [t2].[Phone], [t2].[Fax],
       [t4].[test] AS [test2], [t4].[EmployeeID] AS [EmployeeID2], [t4].[LastName],
       [t4].[FirstName], [t4].[Title], [t4].[TitleOfCourtesy], [t4].[BirthDate],
       [t4].[HireDate], [t4].[Address] AS [Address2], [t4].[City] AS [City2],
       [t4].[Region] AS [Region2], [t4].[PostalCode] AS [PostalCode2],
       [t4].[Country] AS [Country2], [t4].[HomePhone], [t4].[Extension], [t4].[Notes],
       [t4].[ReportsTo],
       [t6].[test] AS [test3], [t6].[ShipperID], [t6].[CompanyName] AS [CompanyName2],
       [t6].[Phone] AS [Phone2], [t6].[LastEditDate], [t6].[CreationDate]
FROM [dbo].[Orders] AS [t0]
LEFT OUTER JOIN (
    SELECT 1 AS [test], [t1].[CustomerID], [t1].[CompanyName], [t1].[ContactName],
           [t1].[ContactTitle], [t1].[Address], [t1].[City], [t1].[Region],
           [t1].[PostalCode], [t1].[Country], [t1].[Phone], [t1].[Fax]
    FROM [dbo].[Customers] AS [t1]
) AS [t2] ON [t2].[CustomerID] = [t0].[CustomerID]
LEFT OUTER JOIN (
    SELECT 1 AS [test], [t3].[EmployeeID], [t3].[LastName], [t3].[FirstName],
           [t3].[Title], [t3].[TitleOfCourtesy], [t3].[BirthDate], [t3].[HireDate],
           [t3].[Address], [t3].[City], [t3].[Region], [t3].[PostalCode],
           [t3].[Country], [t3].[HomePhone], [t3].[Extension], [t3].[Notes],
           [t3].[ReportsTo]
    FROM [dbo].[Employees] AS [t3]
) AS [t4] ON [t4].[EmployeeID] = [t0].[EmployeeID]
LEFT OUTER JOIN (
    SELECT 1 AS [test], [t5].[ShipperID], [t5].[CompanyName], [t5].[Phone]
    FROM [dbo].[Shippers] AS [t5]
) AS [t6] ON [t6].[ShipperID] = [t0].[ShipVia]
```

The query to return successive pages is even more complex. It's obvious that a great deal of effort went into designing the query pipeline's expression tree for eager-loading EntityRef objects.

Setting the `DataLoadOptions` in the `OnCreated()` event handler is required when you instantiate the `DataContext` with a `LinqDataSource` or `ObjectDataSource` control. The `OnCreated()` event fires before `DataSource` controls issue the first LINQ query, so it's the best place to set or invoke custom `DataContext` property values or methods. The `OnCreated()` event also is the place to add `DataContext` customization code that applies to all DAL clients.

## Databinding Windows Form Controls to Entities

Binding `DataContext.Table< TEntity >` collections to data-enabled Windows form components and controls uses a process similar to that for `DataSets`. The primary difference is the requirement to add an Object Data Source for the collection. Controls bound directly to `DataContext.Table< TEntity >` collections are read-write for modifying entities, but you can't add or remove them. Adding an intermediary `BindingSource` component enables adding and removing entities. The UI control (`DataGridView`, list, or text box control) buffers the collection. If you want to cache the collection from `Table< TEntity >` collection or a LINQ query over it, invoke the `Table< TEntity >.ToList()` or `IEnumerable< T >.ToList()` method to populate a generic `List< T >`.

You can create hierarchical master-child[-grandchild] details-style or grid-based data editing forms from a single entity by populating subforms from the master entity's `EntitySet`(s) and subsets. This process emulates populating child grids by specifying foreign-key relationships of the source `DataSet`'s  `DataTables` as their data source.

*The `LinqToSqlDataboundControlsCS.sln` and `LinqToSqlDataboundControlsVB.sln` sample projects for the following sections are located in the `\WROX\ADONET\Chapter05\CS\ \LinqToSqlDataboundControls` and ... \VB\ `LinqToSqlDataboundControls` folders.*

### Autogenerating the Obligatory Hierarchical Data Editing Form

Following are the basic steps for creating a three-level, hierarchical data display and update form based on an Object Data Source bound to a `Table< Customers >` collection from the Northwind sample database:

1. Create a Windows form project and then generate a LINQ to SQL class or class library named `Northwind` from the Northwind database with at least `Customer`, `Order`, and `Order_Detail` entities as described in the earlier "Mapping Tables to Entity Sets with the LINQ to SQL O/R Designer" section.
2. Choose Data, Add New Data Source to start the Data Source Configuration Wizard, select Object as the Data Source Type, and click Next.
3. Expand the two `ProjectName` nodes in the tree view and select the entity to serve as the data source for the components or controls, `Customer` for this example (see Figure 5-12).

## Part III: Applying Domain-Specific LINQ Implementations



Figure 5-12

4. Click Next and Finish to add the new Customer data source to the Data Sources window and dismiss the wizard.
5. Open the Data Sources window and drag the Customer node to the empty form to add a CustomerBindingNavigator and CustomerDataGridView to the form, and CustomerBindingNavigator and CustomerBindingSource icons to the tray. Adjust the size of the form and size and position of the DataGridView.
6. Expand the Data Sources window's Customer node to expose its Orders EntitySet subnode, and drag the Orders subnode to the form to add an OrdersBindingSource and OrdersDataGridView to the form. The OrdersBindingSource's DataSource autogenerated property value is the CustomerBindingSource and itsDataMember property value is "Orders."
7. Expand the Orders subnode to display the Order\_Details subnode, and drag it to the form to add the Order\_DetailsBindingSource and Order\_DetailsDataGridView. The Order\_DetailsBindingSource's DataSource property value is the OrdersBindingSource and itsDataMember property value is "Order\_Details". The Data Sources window and your form appear as shown in Figure 5-13.

## Chapter 5: Using LINQ to SQL and the LinqDataSource

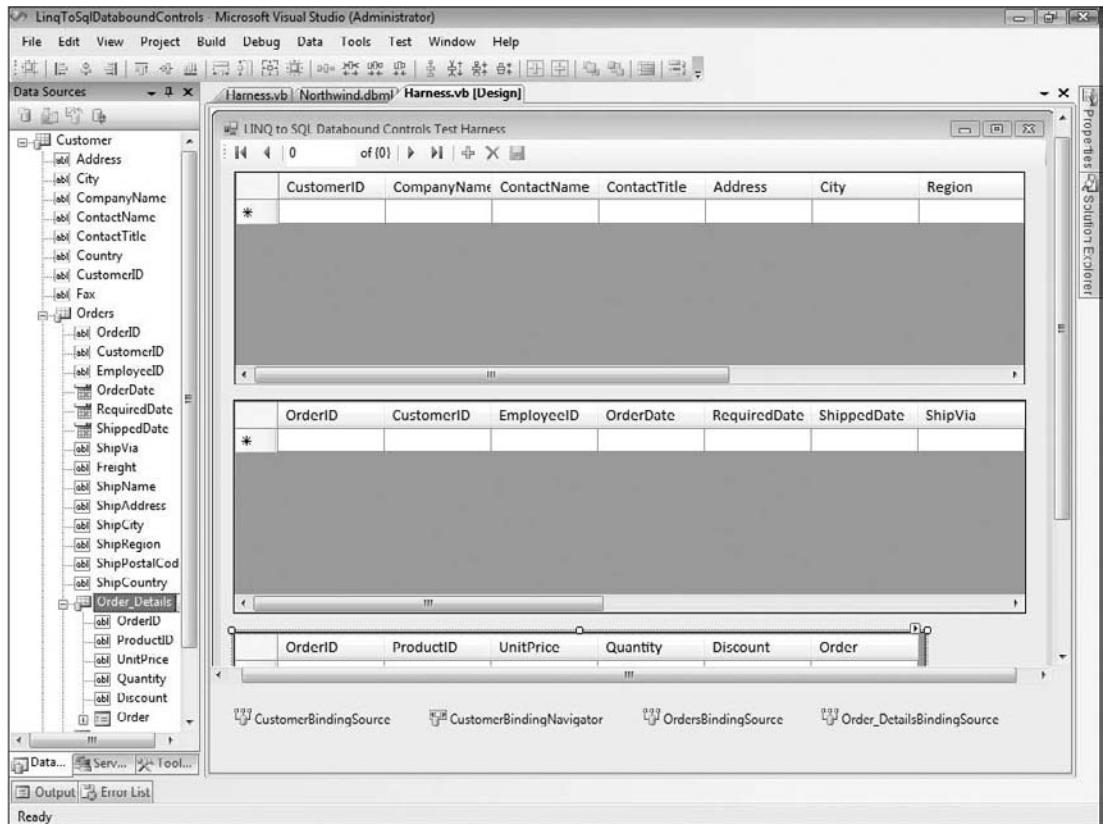


Figure 5-13

8. DataGridViews don't support displaying values from EntityRefs, so edit the OrdersDataGridView's columns to delete Customer and remove the Order column from the Order\_DetailsDataGridView.
9. Declare a NorthwindDataContext module-level (form-level) variable, as follows:

### C# 3.0

```
NorthwindDataContext dcNwind;
```

### VB 9.0

```
Private dcNwind As NorthwindDataContext
```

## Part III: Applying Domain-Specific LINQ Implementations

---

- 10.** Double-click an empty area of the form to generate a `Form_Load` event handler and add the following two lines of code to instantiate the `NorthwindDataContext` and set it as the `CustomerBindingSource.DataSource` property value:

### C# 3.0

```
dcNwind = new NorthwindDataContext();
customerBindingSource.DataSource = dcNwind.Customers;
```

### VB 9.0

```
dcNwind = New NorthwindDataContext()
CustomerBindingSource.DataSource = dcNwind.Customers
```

Press F5 to build and run the project, and populate the three `DataGridViews`. Select a few different Customer and Order entities to verify that the associations are mapped as expected.

## **Persisting Entity Edits and Collection Modifications**

To persist entity edits and collection modifications to the data store, right-click the `CustomerBindingNavigator`'s Save Data button, choose Enabled, double-click the button to add a `CustomerBindingNavigatorSaveItem_Click` event handler. Change `dcNwind` to a form-level variable and add this code to complete pending edits and invoke the `NorthwindDataContext.SubmitChanges()` method:

### C# 3.0

```
order_DetailsBindingSource.EndEdit();
ordersBindingSource.EndEdit();
customerBindingSource.EndEdit();
dcNwind.SubmitChanges();
```

### VB 9.0

```
Order_DetailsBindingSource.EndEdit()
OrdersBindingSource.EndEdit()
CustomerBindingSource.EndEdit()
dcNwind.SubmitChanges()
```

Figure 5-14 shows edits in process for an order for a customer previously added to the Northwind database.

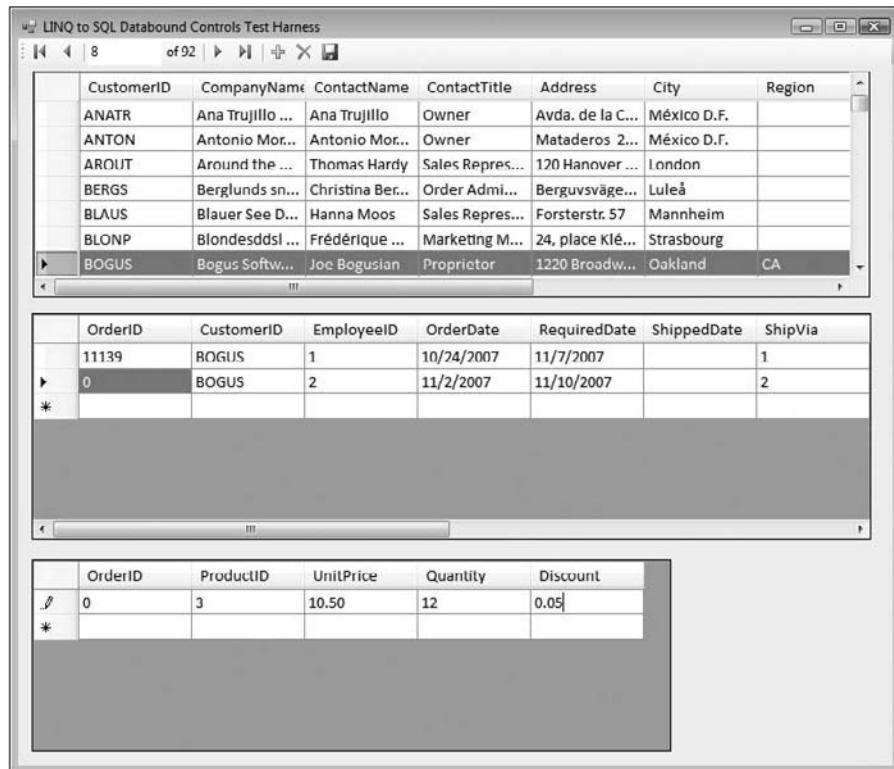


Figure 5-14

### ***Adding Members to a Collection with a Surrogate, Autoincrementing Primary Key***

The Orders table's OrderID column has the `int identity` data type so `Order.OrderID`'s Auto Generated property value defaults to True and the Auto-Sync defaults to OnInsert (other choices are Never, OnUpdate, and Always). These settings cause the T-SQL `INSERT` statement generated by the expression tree to issue a `SELECT SCOPE_IDENTITY() FROM Orders` query to return the identity value for the new `Order` entity and set the new `Order_Detail.OrderID` value.

By default, rows added to a `DataGridView` that's bound to a `Table< TEntity >` with an autoincrementing primary key display 0 as the temporary primary key value; child rows display 0 as foreign-key values also. (This contrasts with `DataSets`, which start with a temporary primary key of -1 and decrement the values for added rows and foreign-key values of child rows.) When you invoke the `SubmitChanges()` method, the `DataContext.ChangeProcessor()` object updates the primary key values of added rows and associated child rows.

## Part III: Applying Domain-Specific LINQ Implementations

### **Deleting Members with a Dependent EntitySet from a Collection**

Northwind's tables are unrealistic because foreign-key values are nullable unless they're members of a composite primary key. In most tables, only one non-key field per table isn't nullable, such as CompanyName for the Customers and Suppliers tables, ProductName for the Products table, and FirstName and LastName for the Employees table. When you delete one or more Order\_Detail rows from the DataGridView, alone or in conjunction with deleting an Order row, submitting the changes causes the `DataContext`'s `ChangeTracker` to throw the exception: "An attempt was made to remove a relationship between a[n] Order and a[n] Order\_Detail. However, one of the relationship's foreign keys (`Order_Detail.OrderID`) cannot be set to null." The exception occurs because the `Order_Detail.OrderID` field is a member of the table's composite primary key; the other member is `Order_Detail.ProductID`.

### **Specifying SQL Server Cascading Deletions**

No Action is the default `DeleteRule` and `UpdateRule` property values for the `INSERT` and `UPDATE` Specification in SQL Server's Foreign Key Relationships dialog for all Northwind tables; alternative selections are Cascade, Set Null, and Set Default. The alternative choices correspond to adding `ON DELETE CASCADE`, `ON DELETE SET NULL` or `ON DELETE SET DEFAULT` clauses to the T-SQL `CREATE | ALTER TABLE` instruction.

*To open the Foreign Key Relationships dialog in Server Explorer, right-click the TableName node, choose Open Table Definition, and then click the toolbar's Relationships button. In SQL Server Management Studio [Express], expand the dependent dbo.TableName's node in Object Explorer, expand the Keys node, right-click the FK\_Relation\_Name, and choose Modify.*

If you have `ALTER TABLE` permission for the database, you can prevent the preceding exception by selecting Cascade, as shown in Figure 5-15, and saving your changes.

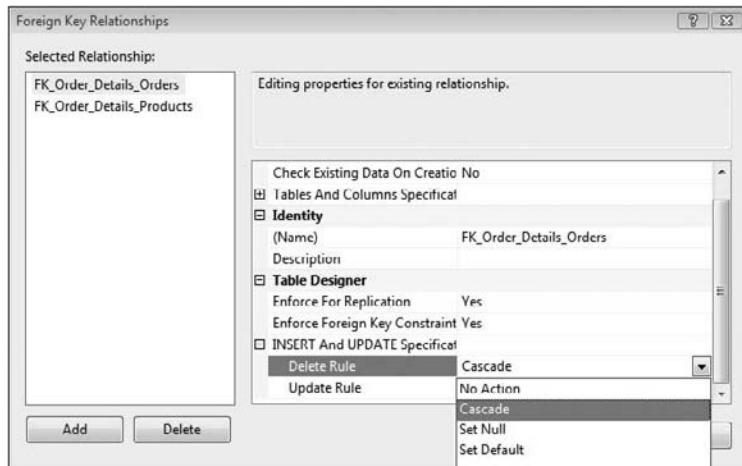


Figure 5-15

## Chapter 5: Using LINQ to SQL and the LinqDataSource

This change generates the following T-SQL Data Definition Language (DDL) statement:

```
ALTER TABLE [dbo].[Order Details] WITH NOCHECK ADD CONSTRAINT [FK_Order_Details_Orders] FOREIGN KEY([OrderID])
REFERENCES [dbo].[Orders] ([OrderID])
ON DELETE CASCADE
```

To eliminate problems deleting Customer entities that have dependent Orders EntityRef(s), make the same change to the FK\_Orders\_Customers relationship.

*The DataSet Designer's Relationships dialog is similar to SQL Server's Foreign Key Relationships dialog. The default setting is the Foreign Key Only, but it also has a Both Relation and Foreign Key Constraints option. Update Rule and Delete Rule have a None, Cascade, Set Null, or Set Default choice.*

### Adding the DeleteOnNull Attribute for Value-Type Association Keys

If you don't have the required ALTER TABLE permission, you can add a DeleteOnNull="true" attribute name-value pair to the \*.dbml file's <Association> node for the foreign-key component of the relationship. This approach sets the Orders.CustomerID property and table column values to NULL, which removes the rows from the association, but doesn't remove these now-orphaned rows from the persistence store.

*Leaving orphaned rows in relational databases isn't an accepted practice. Apparently, the LINQ to SQL team took this approach as a result of a concern that deleted dependent objects might need to be retained to satisfy other associations representing a many:many (m:n) relationship. However, LINQ to SQL doesn't support m:n relationships directly.*

Following is the node of Northwind.dbml file for the Order\_Order\_Detail relationship with the attribute added:

```
<Association Name="Order_Order_Detail" Member="Order" ThisKey="OrderID"
Type="Order" IsForeignKey="true" DeleteOnNull="true" />
```

*You must edit the \*.dbml file in the XML Editor (not the XML Editor with Encoding) to add the required attribute; it's not present in the association's property sheet. When you rebuild and run your project you'll probably find that the DataContext class's default parameterless constructor has disappeared. You can reconstruct it with the following code or add the parameter for the NorthwindConnectionString to the constructor method call in the Form\_Load event handler.*

### C# 3.0

```
public NorthwindDataContext() :
    base(Properties.Settings.Default.NorthwindConnectionString, mappingSource)
{
    OnCreated();
}
```

### VB 9.0

```
Public Sub New()
    MyBase.New(My.Settings.NorthwindConnectionString, mappingSource)
    OnCreated()
End Sub
```

## Part III: Applying Domain-Specific LINQ Implementations

---

Adding a `DeleteOnNull="true"` attribute after the `HasForeignKey="true"` attribute of the `<Association Name="CustomerOrder ... />` node throws this exception: "Error DBML1055: The `DeleteOnNull` attribute of the Association element 'Customer\_Order' can only be true for singleton association members mapped to non-nullable foreign key columns."

### Deleting Dependent Records with `UserDeletingRows` Event Handlers

As a rule, it's a better approach to enable cascading deletions on dependent rows in the database unless you need to retain orphaned records by marking rows deleted instead of removing them. However, some DBAs won't permit use of cascading deletions in "their" databases because it's easy to accidentally delete many dependent rows by deleting the parent row.

The LINQ query expressions in the following `DataGridView_UserDeletingRows` event handlers walk the associations to delete the appropriate dependent rows when the user selects a row and presses the Delete key:

#### C# 3.0

```
private void customerDataGridView_UserDeletingRow(object sender,
    System.Windows.Forms.DataGridViewRowCancelEventArgs e)
{
    var custDelete = dcNwind.Customers.Where(c => c.CustomerID ==
        e.Row.Cells[0].Value.ToString()).FirstOrDefault();
    var ordDeleteColl = custDelete.Orders.Where(o => o.CustomerID ==
        custDelete.CustomerID).AsEnumerable();
    foreach (Order ordDelete in ordDeleteColl)
    {
        var dtlsDelete = ordDelete.Order_Details.Where(d => d.OrderID ==
            (int)order_DetailsDataGridView.CurrentRow.Cells[0].Value).AsEnumerable();
        dcNwind.Order_Details.RemoveAll(dtlsDelete);
        dcNwind.Orders.Remove(ordDelete);
    }
}

private void ordersDataGridView_UserDeletingRow(object sender,
    System.Windows.Forms.DataGridViewRowCancelEventArgs e)
{
    var custDelete = dcNwind.Customers.Where(c => c.CustomerID ==
        customerDataGridView.CurrentRow.Cells[0].Value.ToString()).FirstOrDefault();
    var ordDelete = custDelete.Orders.Where(o => o.OrderID ==
        (int)e.Row.Cells[0].Value).FirstOrDefault();
    var dtlsDelete = ordDelete.Order_Details.Where(d => d.OrderID ==
        (int)order_DetailsDataGridView.CurrentRow.Cells[0].Value).AsEnumerable();
    dcNwind.Order_Details.RemoveAll(dtlsDelete);
    dcNwind.Orders.Remove(ordDelete);
}

private void order_DetailsDataGridView_UserDeletingRow(object sender,
    System.Windows.Forms.DataGridViewRowCancelEventArgs e)
{
    var custDelete = dcNwind.Customers.Where(c => c.CustomerID ==
        customerDataGridView.CurrentRow.Cells[0].Value.ToString()).FirstOrDefault();
```

## Chapter 5: Using LINQ to SQL and the LinqDataSource

---

```
var ordDelete = custDelete.Orders.Where(o => o.OrderID ==  
    (int)ordersDataGridView.CurrentRow.Cells[0].Value).FirstOrDefault();  
var dtlDelete = ordDelete.Order_Details.Where(d => d.OrderID ==  
    (int)e.Row.Cells[0].Value).FirstOrDefault();  
dcNwind.Order_Details.Remove(dtlDelete);  
}
```

### VB 9.0

```
Private Sub CustomerDataGridView_UserDeletingRow(ByVal sender As Object, _  
    ByVal e As System.Windows.Forms.DataGridViewRowCancelEventArgs) _  
    Handles CustomerDataGridView.UserDeletingRow  
Dim custDelete = dcNwind.Customers.Where(Function(c) c.CustomerID = _  
    e.Row.Cells(0).Value.ToString()).FirstOrDefault()  
Dim ordsDelete = custDelete.Orders.Where(Function(o) o.CustomerID = _  
    custDelete.CustomerID).AsEnumerable()  
For Each ordDelete In ordsDelete  
    Dim dtlsDelete = ordDelete.Order_Details.Where(Function(d) d.OrderID = _  
        CInt(Order_DetailsDataGridView.CurrentRow.Cells(0).Value)).AsEnumerable()  
    dcNwind.Order_Details.RemoveAll(dtlsDelete)  
    dcNwind.Orders.Remove(ordDelete)  
Next  
End Sub  
  
Private Sub OrdersDataGridView_UserDeletingRow(ByVal sender As Object, _  
    ByVal e As System.Windows.Forms.DataGridViewRowCancelEventArgs) _  
    Handles OrdersDataGridView.UserDeletingRow  
Dim custDelete = dcNwind.Customers.Where(Function(c) c.CustomerID = _  
    CustomerDataGridView.CurrentRow.Cells(0).Value.ToString()).FirstOrDefault()  
Dim ordDelete = custDelete.Orders.Where(Function(o) o.OrderID = _  
    e.Row.Cells(0).Value).FirstOrDefault()  
Dim dtlsDelete = ordDelete.Order_Details.Where(Function(d) d.OrderID = _  
    CInt(Order_DetailsDataGridView.CurrentRow.Cells(0).Value)).AsEnumerable()  
dcNwind.Order_Details.RemoveAll(dtlsDelete)  
dcNwind.Orders.Remove(ordDelete)  
End Sub  
  
Private Sub Order_DetailsDataGridView_UserDeletingRow(ByVal sender As Object, _  
    ByVal e As System.Windows.Forms.DataGridViewRowCancelEventArgs) _  
    Handles Order_DetailsDataGridView.UserDeletingRow  
Dim custDelete = dcNwind.Customers.Where(Function(c) c.CustomerID = _  
    CustomerDataGridView.CurrentRow.Cells(0).Value.ToString()).FirstOrDefault()  
Dim ordDelete = custDelete.Orders.Where(Function(o) o.OrderID = _  
    CInt(OrdersDataGridView.CurrentRow.Cells(0).Value)).FirstOrDefault()  
Dim dtlDelete = ordDelete.Order_Details.Where(Function(d) d.OrderID = _  
    CInt(e.Row.Cells(0).Value)).FirstOrDefault()  
dcNwind.Order_Details.Remove(dtlDelete)  
End Sub
```

Other approaches to deleting grid rows and then updating the persistence store can save one or two lines of code, but the preceding design would be valid for *m:n* associations, if present, as well as *1:n* associations.

## Part III: Applying Domain-Specific LINQ Implementations

---

Other than minor differences in autogenerated hierarchical editing forms, temporary primary key values, and handling cascade deletions, bound Windows form controls bound to LINQ to SQL data sources behave identically to those bound to strongly typed DataSet data sources.

## Summary

LINQ to SQL object graphs will become a major contender to DataSets as the preferred DAL for all but the most trivial data-intensive ASP.NET Web sites. LINQ to SQL's graphical O/R Designer makes the transition from the row-column *weltanschauung* of ADO.COM and early ADO.NET implementations to truly object-oriented DALs. LINQ to SQL DALs enable implementing the DataMapper application pattern, as well as Unit of Work, Metadata Mapping, Identity Map, Foreign Key Mapping, and Lazy Loading patterns for projects that use SQL Server 200x [Express] for object persistence. Hopefully, the ADO.NET team will rectify the current lack of support for disconnected change tracking long before the next VS release. You can easily substitute stored procedures for dynamic T-SQL INSERT, UPDATE, and DELETE queries. The ADO.NET Entity Framework, Entity Data Model, Entity SQL, and LINQ to Entities implementation promises to flatten the path to enterprise-scale, object-oriented DALs with more entity inheritance models and database vendor/product-independent EntityClient implementations.

*An informal survey of 4,899 .NET developers conducted in October 2008 by Microsoft developer evangelist Scott Hanselman reported that 1,887 respondents used DataSets in their projects and 1,734 respondents used LINQ to ADO. You can view responses for all 14 of the survey's .NET 2008 SP1 features at <http://www.hanselman.com/blog/SurveyRESULTSWhatNETFrameworkFeaturesDoYouUse.aspx>.*

ASP.NET 3.5's new LinqDataSource control simplifies binding LINQ to SQL entity sets to data-enabled Web UI controls, and enables server-side paging and sorting. The Configure Datasource wizard lets you connect to one of your project's DataContexts, select one of its Table< TEntity >s, add a Where clause constraint, and apply a custom sort order to the LINQ query that executes when the page loads. All data-enabled VS Toolbox and third-party UI controls can bind to the LinqDataSet. You can override lazy loading to minimize the frequency of database reconnections by attaching a LoadOptions collection to the DataContext in the DataContext.OnCreate() event handler. If you want to isolate the DataContext from your presentation layer or replace the LINQ query and its T-SQL batch(es) that hydrate an entity set from its backing store, you can substitute an ObjectDataSource for the LinqDataSource control.

LINQ to SQL's DataContext.Table< TEntity > objects can serve as a data source for databound Windows form controls also. The process of autogenerated BindingNavigators, BindingSources, and DataGridViews or text boxes is quite similar to that for the more traditional strongly typed DataSets. To prevent exceptions when deleting entities with populated EntitySets, you must enable SQL Server's cascading deletions feature, manually modify the \*.dbml file or add code to handlers for the DataGridView\_UserDeletingRows event to cascade the deletions on the client.

# 6

## Querying DataTables with LINQ to DataSet

LINQ to DataSet is probably the least used of Visual Studio 2008's four initial Microsoft LINQ implementations. Most .NET developers who've adopted business object architecture and LINQ for new applications substitute `List<T>` or other generic collections for `DataTables` and LINQ to SQL's `DataContext` or the Entity Framework's `ObjectContext` objects for `DataSets` and `TableAdapters`. However, there are millions of lines of VB 8.0 or C# 2.0 or earlier code in data-intensive, production Windows or Web-based .NET projects that use `DataSets`. The in-memory `DataSet` has been Microsoft's preferred relational data manipulation technology since ADO.NET 1.0's arrival with VS 2002's release to manufacturing (RTM) in early 2002.

The ADO.NET team designed a complete infrastructure of wizards and components devoted to codeless or almost codeless, drag-and-drop databinding to basic Windows form controls such as labels and text, combo boxes, and list boxes, as well as complex-bound `DataGridView` controls with `BindingSource` components. The `Configure Data Source Wizard` lets programmers materialize typed `DataSet` data sources from databases, SOAP Web services, or objects with a few mouse clicks. Similarly, ASP.NET developers implemented `SqlDataSource`, `AccessDataSource`, `XmlDataSource`, and `ObjectDataSource` components for binding data to basic HTML controls and specialized `GridView`, `DetailsView`, `FormView`, `DownList`, `ListView`, and `Repeater` server controls.

*This chapter assumes that you're at least acquainted with untyped and typed `DataSet` objects and their descendants: `DataTables`, `TableAdapters`, `DataAdapters`, `DataViews`, `DataRow`s, `DataRowView`s, and so on. The sample code in your \WROX\ADONET\Chapter06 folder's CS and VB subfolders contains code to load `DataSets` with `TableAdapters` connected to the Northwind sample database attached to a local SQL Server 2005+ Express instance. Some examples can update the tables, so you should back up both databases before running those examples.*

## Part III: Applying Domain-Specific LINQ Implementations

---

You can bind `List<T>` entity collections you create by invoking LINQ's `IEnumerable<T>.ToList()` or `IQueryable<T>.ToList()` method directly to the `DataSource` property of `DataGridView` controls, but taking advantage of an intervening `BindingSource` component enables grid-based updates. The same is true for databinding ASP.NET controls, except that one of the `...DataSource` controls is required; you can use the `ObjectDataSource` or ASP.NET 3.5's new `LinqDataSource` to bind generic lists, as described in the "ASP.NET Databinding with the `LinqDataSource` Control" section of Chapter 5. The `LinqDataSource` component enables server-side paging and sorting.

*This book uses the term entity in two ways — as a set or collection of objects having the same attributes or an individual member of such a set of objects. In this chapter, entity means the latter, unless indicated otherwise.*

## Comparing DataSets and DataContexts

Deciding between the use of relational-oriented `DataSets` or object-oriented data entities in new Visual Studio 2008 projects often isn't easy, especially if you've accumulated expertise with programming `DataSets` and their sub-objects, methods, and properties. Programming business objects is becoming the dominant approach to writing enterprise-scale applications for a wide range of industries. The migration from tabular, relational data to objects has occurred for a variety of reasons, not the least of which was the early ascendancy of Java components in the middle tier. The ability to encapsulate business rules as behaviors in business objects is another of the primary incentives for the transition from manipulating individual data rows and columns to instantiating business objects and setting their property values within the constraints of their business rules.

`DataSets` have many characteristics in common with collections of related entities, especially collections created by object/relational mapping (O/RM) tools, such as LINQ to SQL and the Entity Framework (EF). All three technologies store their collections in memory. A `DataSet`'s collection of related `DataTable` objects are disconnected from their relational data store after populating them from the data stored; the `SqlDataAdapter` opens and closes the database connection automatically as required. Performing updates on the underlying physical tables also involves creating a temporary database connection. Similarly, LINQ to SQL and EF entity collections are disconnected by default except while querying or updating their persistence store.

The following table compares the more important objects and features of typed `DataSets` and `DataContexts`.

DataSet Object or Feature	DataContext Object or Feature
<code>DataTable</code> of typed <code>DataSet</code> .	<code>Generic Table&lt;T&gt;</code> , a collection of <code>entity&lt;T&gt;s</code> .
<code>DataRow</code> of typed <code>DataTable</code> .	<code>Generic entity&lt;T&gt;</code> .
Relations of typed <code>DataSet</code> are enforced by <code>DataTable.Constraints</code> .	Associations of <code>entity&lt;T&gt;s</code> represent object relationships.
Many:1 relationship between typed <code>DataTables</code> .	<code>EntityRef&lt;TEntity&gt;</code> association between <code>Table&lt;T&gt;s</code> .

## Chapter 6: Querying DataTables with LINQ to DataSet

DataSet Object or Feature	DataContext Object or Feature
1:many relationship between typed DataTables.	EntitySet< TEntity > association between Table< T >s.
DataTable's primary key field stores row identity.	System.Data.Linq.IdentityManager maintains row identity and keeps only one copy of the entity in memory.
DataTable's foreign keys relate DataTables.	Surrogate foreign keys aren't a valid object property but are useful to regenerate associations with other entities.
DataViews can be filtered and sorted, but DataTables don't support joins.	All LINQ standard query operators apply to Table< T > data sources, including Join().
DataRow s store DataRow.RowState, original values and current values; can be persisted as XML DiffGrams.	System.Data.Linq.StandardChangeTracker object stores entity state and original values; can be persisted to database only.
DataViews enable implementing IBindingListView for advanced filters and user-selected sorting in grid controls.	Generic List< T >s implement IList, which doesn't support user-selected sorts in grid controls.
Updates to tables require applying the DataTable.GetChanges() method for inserts, updates, and deletions, and then invoking the DataSet.SaveChanges() method*.	A single DataContext.SubmitChanges() method call persists all changes to the data store in the appropriate order to maintain referential integrity.
DataSets offer better overall performance than LINQ-based business objects, especially with a large number of rows.	Compiling parameterized LINQ queries minimizes the performance gap.
DataSets accommodate any data source that has an OLE DB driver or a .NET managed data provider.	LINQ to SQL is limited to SQL Server 200x [Express] and SQL Server Compact v3.5 as its data sources; Entity Framework supports most popular databases with third-party managed data providers.
DataSets are fully serializable and are well suited to service-oriented projects that use the Windows Communication Foundation (WCF) for n-tier deployment.	DataContext and Table< T > are <i>not</i> serializable; List< Entity< T > > is serializable with the DataContractSerializer if no EntityRef< TEntity > associations exist.

\*The ADO.NET 3.5 SaveChanges() method's new Hierarchical Update feature uses a TableAdapterManager to automate the update process by adding code to a project's typed DataSet class.

The last two rows of the preceding table might determine the outcome of your decision or affect your ability to use LINQ at all. If you opt for business objects and substitute an O/RM tool that's database-agnostic but doesn't have built-in LINQ compatibility, you won't be able to take advantage of LINQ's powerful query capability. Fortunately, a decision to stick with tried-and-true ADO.NET DataSets doesn't prevent you from taking advantage of LINQ queries.

# Exploring LINQ to DataSet Features

Your primary motives for using LINQ to DataSet queries are likely to be creating joins between DataTables, which conventional DataSets don't support, and replacing the cryptic, string-based filter expressions for generating and sorting DataViews such as `DataView.Sort = "OrderDate DESC"` and `DataView.RowFilter = "OrderDate <= '1/1/2008'"`. If you have a typed DataSet, the compiler will type-check your queries. You also can generate reports from DataTables or DataViews with grouping and aggregate SQOs. LINQ can greatly expand the capabilities of existing DataSet-based projects with only a minor increase in resource overhead.

The `System.Data.DataSetExtensions` namespace in `\Program Files\Reference Assemblies\Microsoft\Framework\v3.5\System.Data.DataSetExtensions.dll` contains the following LINQ to DataSet classes:

- ❑ `TypedTableBase<T>` is a new generic abstract class that inherits from earlier .NET versions' `System.Data.DataTable` class for backward compatibility. `TypedTableBase<T>` implements `IEnumerable<T>` (where `T` is the `DataRow` type) and `IEnumerable` to enable LINQ queries.
- ❑ `TypedTableBaseExtensions` are a set of extension methods for the `AsEnumerable()`, `OrderBy()`, `OrderByDescending()`, `Select()`, and `Where()` SQOs that operate on `EnumerableRowCollection<T>` or `OrderedEnumerableRowCollection<T>` collections.
- ❑ `EnumerableRowCollection<T>` overloads (2) or `OrderedEnumerableRowCollection<T>` collections have a similar set of extension methods plus `ThenBy()` and `ThenByDescending()`, but these methods aren't intended to be called from your code.
- ❑ `DataTableExtensions` include `As DataView<T>` to return a LINQ-enabled typed `DataView` from a `LinqDataView` internal class that derives from `DataView`, `AsEnumerable(DataTable)` returns an `IEnumerable<T>` from untyped DataSets for use in a LINQ query, and three overloads of `CopyToDataTable<T>` extension methods that return a `DataTable` from an `IEnumerable<T>`.
- ❑ `DataRowExtensions` include six overloads of `Field<T>` to provide strongly typed access to unboxed column values by passing the  `DataColumn` object itself, its name as a `String`, or its ordinal as an `int`. The remaining three `Field<T>` overloads also let you specify the `DataRowVersion`. The three `SetField<T>` overloads, which support nullable types, let you set new column values. These extensions are used for untyped data tables.
- ❑ `DataRowComparer` returns a single instance of the `DataRowComparer` class and `DataRowComparer<TRow>` compares two `DataRow` objects by using value-based (not reference-based) comparison.

You must add a reference to the `System.Data.DataSetExtensions` namespace if it's missing, but you don't need a `using/Imports System.Data.DataSetExtensions` directive; the `DataSetExtensions` extend the `System.Data` namespace, which DataSets require. LINQ to DataSet automatically applies the extension methods to ordinary LINQ queries.

Applying the `As DataView()` method to your LINQ query with the internal `LinqDataView` class enables databound grid controls to deliver advanced sorting and filtering with the `IBindingListView` interface, supports indexed row access by the `Find()` and `FindRows()` methods with `FindByKey()` and

`FindRowsByKey()`, and enhances editing capabilities. Figure 6-1 shows the `DataSetExtensions` namespace expanded and the `LinqDataView` class disassembled in Red gate's (formerly Lutz Roeder's) .NET Reflector program. The later "Creating LinqDataViews for DataBinding" section demonstrates the specific benefits of binding grid controls to an `IEnumerable.AsDataView()` data source.

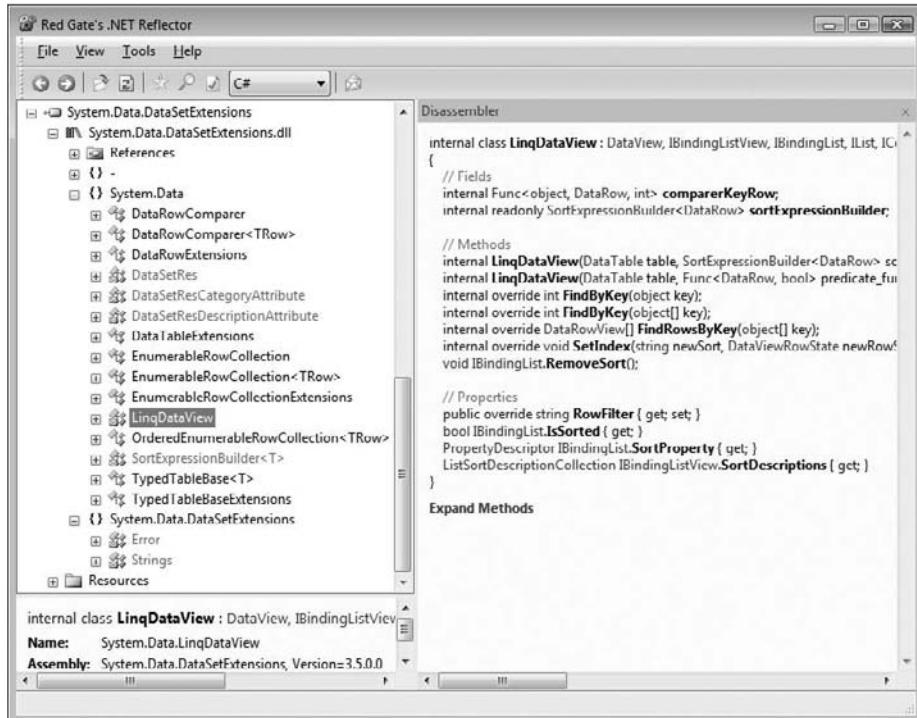


Figure 6-1

## Running Read-Only LINQ to DataSet Queries

Simple, read-only reports in DataGridViews with data sources having complex constraints are a common requirement in data-centric projects. For example, marketing managers often want fast access to sales figures for specified products in a particular country or area. The `UntypedAndTypedDataSets` sample projects populate a DataGridView by setting its `DataSource` property to a generic `List<AnonymousType>` with the `IEnumerable.ToList()` method. Users select a product and country from lookup lists populated by the Northwind Products table and a LINQ query against the `Country` field of the `Customers` table followed by clicking a button to commit the query. Figure 6-2 shows the UI of the C# version with the default product and country selections.

## Part III: Applying Domain-Specific LINQ Implementations

	CustomerID	CompanyName	OrderID	ShippedDate	ProductName	Quantity
1	RATTIC	Rattlesnake Canyon Grocery	11077		Chang	24
2	SAVEA	Save-a-lot Markets	11030	4/27/2007	Chang	100
3	RATTIC	Rattlesnake Canyon Grocery	10852	1/30/2007	Chang	15
4	SAVEA	Save-a-lot Markets	10722	11/4/2006	Chang	3
5	SAVEA	Save-a-lot Markets	10714	10/27/2006	Chang	30
6	WHITC	White Clover Markets	10504	4/18/2006	Chang	12
7	WHITC	White Clover Markets	10469	3/14/2006	Chang	40
8	SAVFA	Save-a-lot Markets	10440	2/28/2006	Chang	45
9	SAVEA	Save-a-lot Markets	10393	1/3/2006	Chang	25

Figure 6-2

The two samples have a check box that defaults to LINQ `join...in...on...equals/Join...In...On...Equals` expressions but enables executing an alternative syntax that sets up DataRelations between the tables and replaces `Orders`, `Order_Details`, and `Products` DataTables with `FK_Customers_Orders`, `FK_Orders_Order_Details`, and `FK_Order_Details_Products` DataRelations. Working with DataRelations is a more object-oriented process because DataRelations correspond to entity associations but involves substantially more code than referencing DataTables. There's no significant performance difference between the two approaches. Forms with hierarchical grids use DataRelations to provide DataSource property values to BindingSource components.

### Querying Untyped DataSets

Untyped DataSets require you to instantiate a new named `DataSet`; you then add, define and fill each `DataTable` by creating and naming a new `DataAdapter` and applying its `Fill()` method. You also use the `DataAdapter` to update the underlying RDBMS tables after editing the `DataTable`s. Untyped DataSets are lightweights compared with typed `DataSets` that are the subject of a later section and execute faster; this example executes about 30% faster than the corresponding typed version.

Untyped `DataTable`s require late-bound references to `DataRow` fields of the object type with code that specifies their name and data type, as in `IEnumerable<T>.Field<string>[ "CustomerID" ]` or `IEnumerable(Of T).Field(Of Integer)(OrderID)`. `T` is always a `DataRow`, but untyped `DataTable` objects have untyped `DataRow`s.

*Code to create and fill the `DataAdapters` or `TableAdapters` is included in the following initial listings as an aid to readers who aren't `DataSet` experts. For brevity, the remaining listings don't include that code.*

### C# 3.0

```
private void btnUntyped_Click(object sender, EventArgs e)
{
    DataSet dsNwindU = new DataSet();
    string strConn = Properties.Settings.Default.NorthwindConnectionString;
    SqlConnection cnNwind = new SqlConnection(strConn);
    using (cnNwind)
    {
        cnNwind.Open();
        // Fill the untyped DataSet's four DataTables
        string strSQL = "SELECT * FROM Customers";
        SqlDataAdapter daCusts = new SqlDataAdapter(strSQL, cnNwind);
        daCusts.Fill(dsNwindU, "Customers");

        strSQL = "SELECT * FROM Orders";
        SqlDataAdapter daOrders = new SqlDataAdapter(strSQL, cnNwind);
        daOrders.Fill(dsNwindU, "Orders");

        strSQL = "SELECT * FROM [Order Details]";
        SqlDataAdapter daDetails = new SqlDataAdapter(strSQL, cnNwind);
        daDetails.Fill(dsNwindU, "Order_Details");

        strSQL = "SELECT * FROM Products";
        SqlDataAdapter daProds = new SqlDataAdapter(strSQL, cnNwind);
        daProds.Fill(dsNwindU, "Products");
        cnNwind.Close();
    }

    FillCombos(dsNwindU.Tables["Products"], dsNwindU.Tables["Customers"]);

    // Create the relationships - Orders:Customers
    DataColumn dcParent = dsNwindU.Tables["Customers"].Columns["CustomerID"];
    DataColumn dcChild = dsNwindU.Tables["Orders"].Columns["CustomerID"];
    DataRelation drOrders = new DataRelation("FK_Orders_Customers",
        dcParent, dcChild);
    dsNwindU.Relations.Add(drOrders);

    // Order_Details:Orders
    dcParent = dsNwindU.Tables["Orders"].Columns["OrderID"];
    dcChild = dsNwindU.Tables["Order_Details"].Columns["OrderID"];
    DataRelation drDetails = new DataRelation("FK_Order_Details_Orders",
        dcParent, dcChild);
    dsNwindU.Relations.Add(drDetails);

    // Order_Details:Products
    dcParent = dsNwindU.Tables["Products"].Columns["ProductID"];
    dcChild = dsNwindU.Tables["Order_Details"].Columns["ProductID"];
    DataRelation drProds = new DataRelation("FK_Order_Details_Products",
        dcParent, dcChild);
    dsNwindU.Relations.Add(drProds);

    if (chkUseJoins.Checked)
    {
        // Execute the query with joins
    }
}
```

## Part III: Applying Domain-Specific LINQ Implementations

---

```
var query = from c in dsNwindU.Tables["Customers"].AsEnumerable()
            join o in dsNwindU.Tables["Orders"].AsEnumerable()
                on c.Field<string>("CustomerID") equals
                o.Field<string>("CustomerID")
            join d in dsNwindU.Tables["Order_Details"].AsEnumerable()
                on o.Field<int>("OrderID") equals
                d.Field<int>("OrderID")
            join p in dsNwindU.Tables["Products"].AsEnumerable()
                on d.Field<int>("ProductID") equals
                p.Field<int>("ProductID")
            where p.Field<string>("ProductName") == cboProduct.Text
                  && c.Field<string>("Country") == cboCountry.Text
            orderby o.Field<int>("OrderID") descending
        select new
        {
            CustomerID = c.Field<string>("CustomerID"),
            CompanyName = c.Field<string>("CompanyName"),
            OrderID = o.Field<int>("OrderID"),
            ShippedDate = o.Field<DateTime?>("ShippedDate"),
            ProductName = p.Field<string>("ProductName"),
            Quantity = d.Field<short>("Quantity")
        };
    // Populate the DataGridView
    dgvDataSets.DataSource = query.ToList();
}
else
{
    // Execute the query based on DataRelations
    var query = from c in dsNwindU.Tables["Customers"].AsEnumerable()
                from o in c.GetChildRows("FK_Orders_Customers")
                from d in o.GetChildRows("FK_Order_Details_Orders")
                from p in d.GetParentRows("FK_Order_Details_Products")
                where p.Field<int>("ProductID") == 2
                      && c.Field<string>("Country") == "USA"
                      && d.Field<short>("Quantity", DataRowVersion.Original) > 1
                      && d.RowState == DataRowState.Unchanged
                orderby o.Field<int>("OrderID") descending
            select new
            {
                CustomerID = c.Field<string>("CustomerID"),
                CompanyName = c.Field<string>("CompanyName"),
                OrderID = o.Field<int>("OrderID"),
                ShippedDate = o.Field<DateTime?>("ShippedDate"),
                ProductName = p.Field<string>("ProductName"),
                Quantity = d.Field<short>("Quantity")
            };
    // Populate the DataGridView
    dgvDataSets.DataSource = query.ToList();
}
}
```

As noted in Chapter 1's brief "Untyped DataSets" section, VB 9.0 can take advantage of VB's bang (!, also called the *pling* or *dictionary lookup*) operator to reduce typing by replacing the *Field* expressions

## Chapter 6: Querying DataTables with LINQ to DataSet

---

with `IEnumerable(Of T)!FieldName` expressions. If `Option Strict` is `On`, `Option Infer` must be `On` to permit VB to use late-bound *ducktyping* of these objects.

*The name ducktyping for dynamic typing in late binding with VB (and in C# with a reference to a third-party DuckTyping library) is based on the adage: "If it walks like a duck and quacks like a duck, it must be a duck." In this case, the compiler infers the types by its current methods rather than by inheritance from a specific class.*

Notice also in the VB example that, unlike C#, VB doesn't require applying the `AsEnumerable()` SQO to untyped `DataTable`s used as query data sources. You must turn off `Option Strict` to use the `bang` operator when specifying in a `Where` clause a field that uses the `=` operator; turning off `Option Strict` isn't a recommended practice.

### VB 9.0

```
Private Sub btnUntyped_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnUntyped.Click
    Dim dsNwindU As New DataSet
    Dim strConn As String = My.Settings.NorthwindConnectionString
    Dim cnNwind As New SqlConnection(strConn)
    Using cnNwind
        cnNwind.Open()
        ' Fill the untyped DataSet's four DataTables
        Dim strSQL As String = "SELECT * FROM Customers"
        Dim daCusts As New SqlDataAdapter(strSQL, cnNwind)
        daCusts.Fill(dsNwindU, "Customers")

        strSQL = "SELECT * FROM Orders"
        Dim daOrders As New SqlDataAdapter(strSQL, cnNwind)
        daOrders.Fill(dsNwindU, "Orders")

        strSQL = "SELECT * FROM [Order Details]"
        Dim daDetails As New SqlDataAdapter(strSQL, cnNwind)
        daDetails.Fill(dsNwindU, "Order_Details")

        strSQL = "SELECT * FROM Products"
        Dim daProds As New SqlDataAdapter(strSQL, cnNwind)
        daProds.Fill(dsNwindU, "Products")

        cnNwind.Close()
    End Using

    ' Create the relationships - Orders:Customers
    Dim dcParent As DataColumn = dsNwindU.Tables("Customers").Columns("CustomerID")
    Dim dcChild As DataColumn = dsNwindU.Tables("Orders").Columns("CustomerID")
    Dim drOrders = New DataRelation("FK_Orders_Customers", _
        dcParent, dcChild)
    dsNwindU.Relations.Add(drOrders)

    ' Order_Details:Orders
    dcParent = dsNwindU.Tables("Orders").Columns("OrderID")
    dcChild = dsNwindU.Tables("Order_Details").Columns("OrderID")
    Dim drDetails = New DataRelation("FK_Order_Details_Orders", _
        dcParent, dcChild)
```

## Part III: Applying Domain-Specific LINQ Implementations

---

```
dsNwindU.Relations.Add(drDetails)

' Order_Details:Products
dcParent = dsNwindU.Tables("Products").Columns("ProductID")
dcChild = dsNwindU.Tables("Order_Details").Columns("ProductID")
Dim drProds As DataRelation = New DataRelation("FK_Order_Details_Products", _
    dcParent, dcChild)
dsNwindU.Relations.Add(drProds)

' Load the combo boxes
FillCombos(dsNwindU.Tables("Products"), dsNwindU.Tables("Customers"))

If chkUseJoins.Checked Then
    ' Use Join expressions
    Dim Query = From c In dsNwindU.Tables("Customers") _
        Join o In dsNwindU.Tables("Orders") _
            On c!CustomerID Equals o!CustomerID _
        Join d In dsNwindU.Tables("Order_Details") _
            On o!OrderID Equals d!OrderID _
        Join p In dsNwindU.Tables("Products") _
            On d!ProductID Equals p!ProductID _
    Where p.Field(Of String)("ProductName") = cboProduct.Text _
        AndAlso c.Field(Of String)("Country") = cboCountry.Text _
    Order By o.Field(Of Int32)("OrderID") Descending _
    Select CustomerID = c!CustomerID, _
        CompanyName = c!CompanyName, _
        OrderID = o!OrderID, _
        ShippedDate = o!ShippedDate, _
        ProductName = p!ProductName, _
        Quantity = d!Quantity
    dgvDataSets.DataSource = Query.ToList()
Else
    ' Execute the query based on DataRelations
    Dim Query = From c In dsNwindU.Tables("Customers"), _
        o In c.GetChildRows("FK_Orders_Customers"), _
        d In o.GetChildRows("FK_Order_Details_Orders"), _
        p In d.GetParentRows("FK_Order_Details_Products") _
    Where p!ProductName.ToString() = cboProduct.Text _
        AndAlso c!Country.ToString() = cboCountry.Text _
        AndAlso d.Field(Of Short)("Quantity", _
            DataRowVersion.Original) > 1 _
        AndAlso d.RowState = DataRowState.Unchanged _
    Order By o!OrderID Descending _
    Select c!CustomerID, c!CompanyName, o!OrderID, _
        o!ShippedDate, p!ProductName, d!Quantity

    dgvDataSets.DataSource = Query.ToList()
End If
dgvDataSets.Columns(0).Width = 150
```

### Customizing Lookup Lists

Lookup lists are another application that LINQ queries can make more efficient. The sample application has two combo boxes that act as lookup lists for products and countries. If you need a sparse lookup list where the list consists of relatively few of the available collection, you can make the data you retrieve for the report's DataTables do double-duty by applying a LINQ query against the data source. The sample application's `FillCombos()` procedure fills the country combo box's list with a `Select` query against the `Customers` DataTable in the following listing. It's likely that there will be orders for all members of the `Products` table, but you need a LINQ query because you can't apply the `Sort()` method to a combo box with a databound list.

#### C# 3.0

```
private void FillCombos(DataTable tblProducts, DataTable tblCountries)
{
    // Fill Products combo with sorted products and default to Chang
    string strProducts = cboProduct.Text;
    var ProdList = from p in tblProducts.AsEnumerable()
                   orderby p.Field<string>("ProductName")
                   select p.Field<string>("ProductName");

    cboProduct.DataSource = ProdList.ToList();

    if (strProducts.Length > 0)
        cboProduct.Text = strProducts;
    else
        cboProduct.Text = "Chang";

    // Fill Customers combo with an unduplicated sorted list of countries
    string strCountry = cboCountry.Text;
    var CtryList = (from c in tblCountries.AsEnumerable()
                   orderby c.Field<string>("Country")
                   select c.Field<string>("Country")).Distinct();

    cboCountry.DataSource = CtryList.ToList();

    if (strCountry.Length > 0)
        cboCountry.Text = strCountry;
    else
        cboCountry.Text = "USA";
}
```

#### VB 9.0

```
Private Sub FillCombos(ByVal tblProducts As DataTable, _
                      ByVal tblCountry As DataTable)
    ' Fill Products combo with sorted products and default to Chang
    Dim ProdList = From p In tblProducts _
                   Order By p.Field(Of String)("ProductName") _
                   Select p.Field(Of String)("ProductName")

    cboProduct.DataSource = ProdList.ToList()

    If Len(strProducts) > 0 Then
        cboProduct.Text = strProducts
    Else
```

## Part III: Applying Domain-Specific LINQ Implementations

---

```
' Default value
cboProduct.Text = "Chang"
End If

' Fill Customers combo with a sorted list of countries
Dim strCountry As String = cboCountry.Text
Dim CtryList = (From c In tblCountry.AsEnumerable() _
    Order By c.Field(Of String)("Country") -
    Select Country = c!Country).Distinct()

cboCountry.DataSource = CtryList.ToList()

If Len(strCountry) > 0 Then
    cboCountry.Text = strCountry
Else
    ' Default value
    cboCountry.Text = "USA"
End If
End Sub
```

The two procedures handle untyped and typed DataTables equally well.

## Querying Typed DataSets

The .NET Common Language Runtime (CLR) and its implement languages have favored strong typing and type-safe coding since Microsoft introduced .NET 1.0 in 2002. Thus Microsoft has promoted strongly typed DataSets by providing developers code-generation utilities to create the XML schema (.xsd) files and DataSet designer classes that define them. For example, the NorthwindTyped.xsd schema file has 637 lines; the NorthwindTyped.Designer.cs file has 5,891 lines and the .vb version has 5,821. Add about 10 lines to fill the DataTables. It takes only about 19 C# or VB instructions to define *and* fill the corresponding untyped DataSet. However, the advantages of type safety often outweigh strongly typed DataSets' performance penalties.

To further encourage beginning developers to adopt typed DataSets, the Visual Studio team added a Data Source window from which programmers can drag DataTables and DataRelations to a form to create master/child/grandchild forms with autogenerated text boxes for the master and DataGridView controls for the child rows. The LINQ to SQL team made good use of this “wizardry” in creating similar Windows forms from the DataContext and its entities.

Typed DataSets let you maintain type-safe code throughout the data objects' lifetimes and enhance the DataTable updating process. You must use typed DataSets to take advantage of the Hierarchical Update feature by automating updates to multiple tables with a TableAdapterManager. If you use the DataRelations (foreign-key) syntax, you lose type safety in your LINQ queries but not in the underlying DataTables, DataRows, and DataRowViews.

### C# 3.0

```
private void btnTyped_Click(object sender, EventArgs e)
{
    // Fill the strongly typed DataSet's TableAdapters
    NorthwindTyped dsNwindT = new NorthwindTyped();
    NorthwindTypedTableAdapters.CustomersTableAdapter dtCusts =
```

## Chapter 6: Querying DataTables with LINQ to DataSet

---

```
        new NorthwindTypedTableAdapters.CustomersTableAdapter();
        dtCusts.Fill(dsNwindT.Customers);
        NorthwindTypedTableAdapters.OrdersTableAdapter dtOrders =
            new NorthwindTypedTableAdapters.OrdersTableAdapter();
        dtOrders.Fill(dsNwindT.Orders);
        NorthwindTypedTableAdapters.ProductsTableAdapter dtProds =
            new NorthwindTypedTableAdapters.ProductsTableAdapter();
        dtProds.Fill(dsNwindT.Products);
        NorthwindTypedTableAdapters.Order_DetailsTableAdapter dtDetails =
            new NorthwindTypedTableAdapters.Order_DetailsTableAdapter();
        dtDetails.Fill(dsNwindT.Order_Details);

        // Fill the combo boxes
        FillCombos(dsNwindT.Products, dsNwindT.Customers);

        if (chkUseJoins.Checked)
        {
            // Execute the query with joins and handle DBNull values of ShippedDate
            var query = from c in dsNwindT.Customers
                        join o in dsNwindT.Orders on c.CustomerID equals o.CustomerID
                        join d in dsNwindT.Order_Details on o.OrderID equals d.OrderID
                        join p in dsNwindT.Products on d.ProductID equals p.ProductID
                        where p.ProductName == cboProduct.Text
                        && c.Country == cboCountry.Text
                        orderby o.OrderID descending
                        select new
                        {
                            c.CustomerID, c.CompanyName, o.OrderID,
                            ShippedDate = o.Field<DateTime?>("ShippedDate"),
                            p.ProductName, d.Quantity
                        };
            dgvDataSets.DataSource = query.ToList();
        }
        else
        {
            // Execute the query based on foreign-key constraints
            var query = from c in dsNwindT.Customers
                        from o in c.GetOrdersRows()
                        from d in o.GetOrder_DetailsRows()
                        from p in d.GetParentRows("FK_Order_Details_Products")
                        where p.Field<string>("ProductName") == cboProduct.Text
                        && c.Country == cboCountry.Text
                        orderby o.OrderID descending
                        select new {
                            c.CustomerID, c.CompanyName, OrderID = o.OrderID,
                            ShippedDate = o.Field<DateTime?>("ShippedDate"),
                            ProductName = p.Field<string>("ProductName"),
                            Quantity = d.Quantity };

            // Populate the DataGridView
            dgvDataSets.DataSource = query.ToList();
        }
    }
```

## Part III: Applying Domain-Specific LINQ Implementations

---

The `ParentTable.GetFieldNameRows()` method lets you refer to strongly typed child `DataRow`s by field name. However, late binding is required for the `DataTable` equivalent of an `EntityRef`, as shown here for the `Products` `DataTable`. Late binding in the projection for `ShippedDate` is required because it must be specified as of the `Nullable<DateTime>` type.

### VB 9.0

```
Private Sub btnTyped_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnTyped.Click
    Using dsNwindT As NorthwindTyped = New NorthwindTyped
        ' Fill the strongly typed DataSet's TableAdapters
        Dim dtCusts As New NorthwindTypedTableAdapters.CustomersTableAdapter
        dtCusts.Fill(dsNwindT.Customers)
        Dim dtOrders As New NorthwindTypedTableAdapters.OrdersTableAdapter
        dtOrders.Fill(dsNwindT.Orders)
        Dim dtProds As New NorthwindTypedTableAdapters.ProductsTableAdapter
        dtProds.Fill(dsNwindT.Products)
        Dim dtDetails As New NorthwindTypedTableAdapters.Order_DetailsTableAdapter
        dtDetails.Fill(dsNwindT.Order_Details)

        ' Load the combo boxes
        FillCombos(dsNwindT.Products, dsNwindT.Customers)

        If chkUseJoins.Checked Then
            ' Use Join expressions
            Dim Query = From c In dsNwindT.Customers _
                Join o In dsNwindT.Orders _
                    On c.CustomerID Equals o.CustomerID _
                Join d In dsNwindT.Order_Details _
                    On o.OrderID Equals d.OrderID _
                Join p In dsNwindT.Products _
                    On d.ProductID Equals p.ProductID _
                Where p.ProductName = cboProduct.Text _
                    AndAlso c.Country = cboCountry.Text _
                Order By o.OrderID Descending _
                Select c.CustomerID, c.CompanyName, o.OrderID, _
                    ShippedDate = o.Field(Of DateTime?)("ShippedDate"), _
                    p.ProductName, d.Quantity

            dgvDataSets.DataSource = Query.ToList()
        Else
            ' Execute the query based on foreign-key constraints (relationships)
            Dim Query = From c In dsNwindT.Customers, _
                o In c.GetOrdersRows, _
                d In o.GetOrder_DetailsRows, _
                p In d.GetParentRows("FK_Order_Details_Products") _
                Where p!ProductName.ToString() = cboProduct.Text _
                    AndAlso c.Country.ToString() = cboCountry.Text _
                Order By o.OrderID Descending _
                Select c.CustomerID, c.CompanyName, o.OrderID, _
                    ShippedDate = o.Field(Of DateTime?)("ShippedDate"), _
                    p!ProductName, d.Quantity
        End If

        ' Populate the DataGridView
    End Using
End Sub
```

```
dgvDataSets.DataSource = Query.ToList()
End If
End Using
dgvDataSets.Columns(0).Width = 150
End Sub
```

# Creating LinqDataViews for DataBinding with AsDataView()

DataTables are a common data source for bound controls in production .NET applications, at least in part because they're the default data sources for databound controls on Windows forms and ASP.NET's SqlDataAdapter and LinqDataSource components are limited to SQL Server 200x connections. VS 2008's main window for ASP.NET projects doesn't include a Data menu choice until you add a typed DataSet from the Add New Item dialog to open an empty DataSet Designer window and add the Data choice. You then drag fields from Server or Database Explorer to the Designer canvas and save the changes to the class file. The Designer adds lines representing DataRelations, as shown in Figure 6-3. The DataSet's TableAdapters appear in the second dialog of the Configure Object Data Source Wizard (see Figure 6-4) and you can select the appropriate Fill or FillBy method in the third dialog.

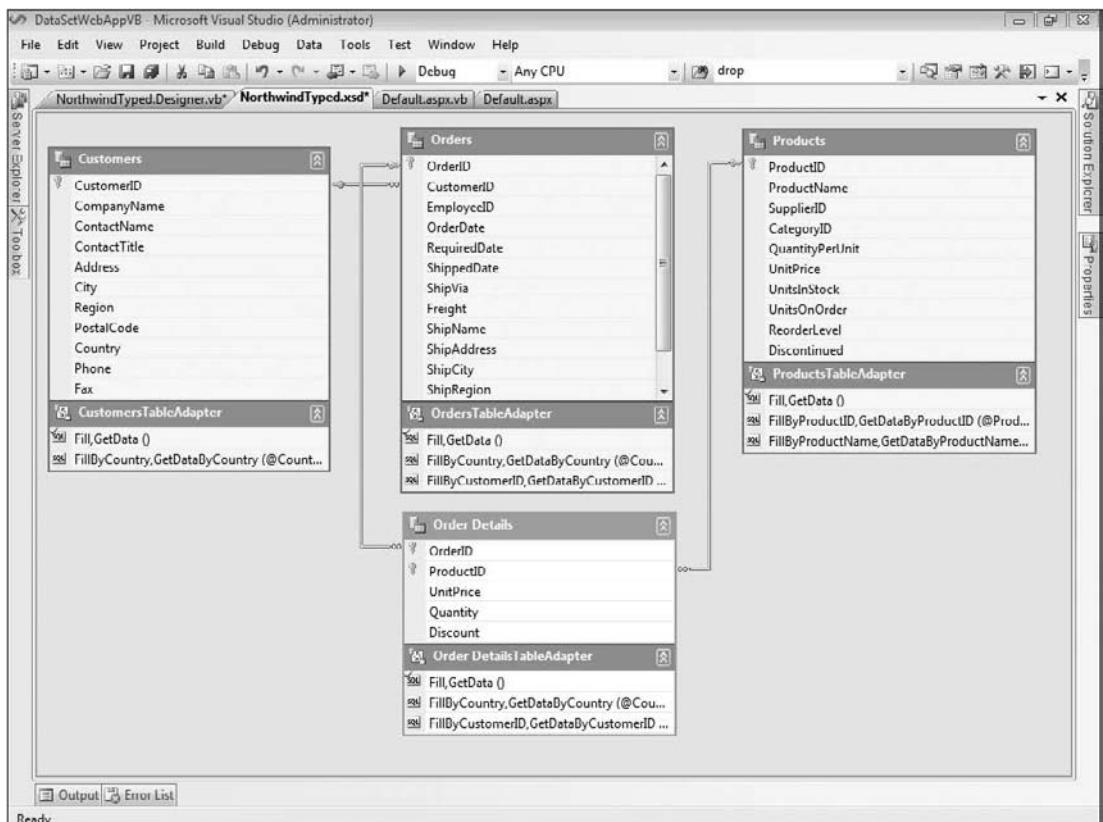


Figure 6-3

## Part III: Applying Domain-Specific LINQ Implementations

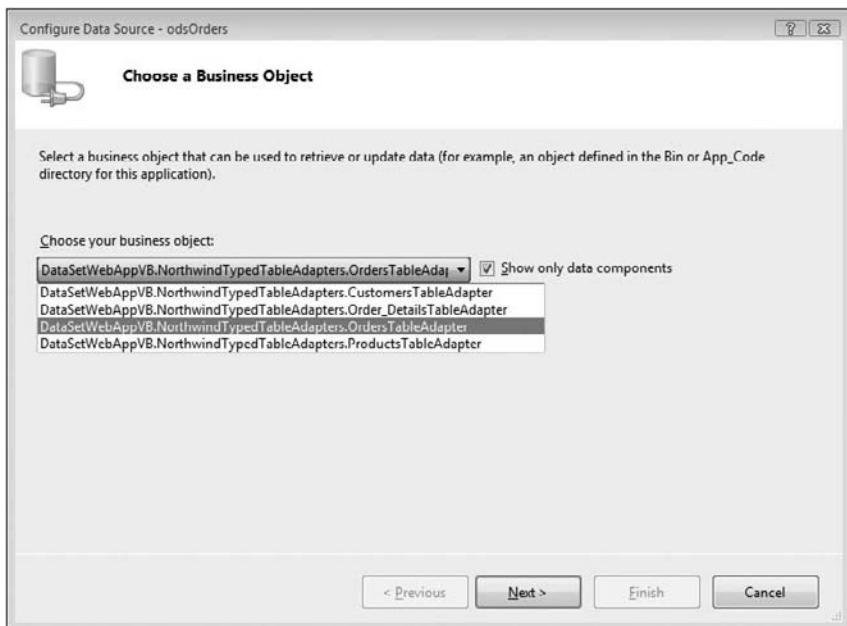


Figure 6-4

*It's possible to substitute an in-memory LINQ to Objects EntitySet for a LINQ to SQL EntitySet as the data source for the LinqDataSource control by adding code to the DataContext .ContextCreating() event handler, as described in a post to the LINQ Project General Forum (see <http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=2574592&SiteID=1>).*

As mentioned in the earlier “Exploring LINQ to DataSet Features” section, the internal `LinqDataSource` class enables LINQ to DataSet to convert a query’s `IEnumerable()` output to a `DataView`, but only if the query doesn’t contain a `select/Select` projection or apply the `Cast()` operator. You can’t apply the `AsDataView()` method to create `DataRow`s from a projection of an anonymous type, so query constraints based on related table values become more complex if you can’t execute the filter expression in the `Where` clause.

For example, the following expression filters an `Orders` `DataTable` for a net extended order amount greater than or equal to a numeric value set in a text box:

### C# 3.0

```
var filteredOrders = from o in dsNwindT.Orders
                     where o.GetOrder_DetailsRows().Sum(d =>
                         .Quantity * d.UnitPrice * (1 - d.Discount)) >=
                         (Decimal)txtValue.Text
                     select o

bsOrders.DataSource = filteredOrders.ToDataView()
```

## Chapter 6: Querying DataTables with LINQ to DataSet

### VB 9.0

```
Dim FilteredOrders = From o In dsNwindT.Orders _
    Where o.GetOrder_DetailsRows().Sum(Function(d) d.Quantity * _
        d.UnitPrice * (1 - d.Discount)) >= _
        CDecimalInt(txtValue.Text) _
    Select o

bsOrders.DataSource = FilteredOrders.ToDataTable()
```

Applying the `ToDataTable()` method enables the `Orders` `BindingSource` component to use the filtered `DataTable`.

*A LINQ query is the only method for applying a filter with calculated expressions to a DataSet.*

The databinding characteristics of `DataViews` generated by LINQ queries vary with the binding component. For example, a `DataView` created by the preceding filter bound to a Windows `DataGridView` control by a `BindingSource` component is updatable. However, the same `DataView` bound to an ASP.NET `GridView` control by an `ObjectDataSource` component is read-only. You can compare the update behavior of filtered and unfiltered `DataViews` with the `DataSetWindowsAppCS` or VB and `DataSetWebAppCS` or VB projects in your \WROX\ADONET\Chapter06\ folder.

The `DataSetWindowsAppCS` and VB (see Figure 6-5) projects let you exchange the `Orders` `DataTable` and the filtered `Orders` `DataView` as the `BindingSource`'s `DataSource` and `DataMember` property values. Updates you make to `DataGridView` cells persist regardless of the `DataSource` because the `DataView` retains its association with the `Orders` `DataTable`. You can prove this by making a change to an `EmployeeID` value with the filter applied, saving the change, removing the filter, refreshing the grid, and verifying the change persisted to the database's `Orders` table.

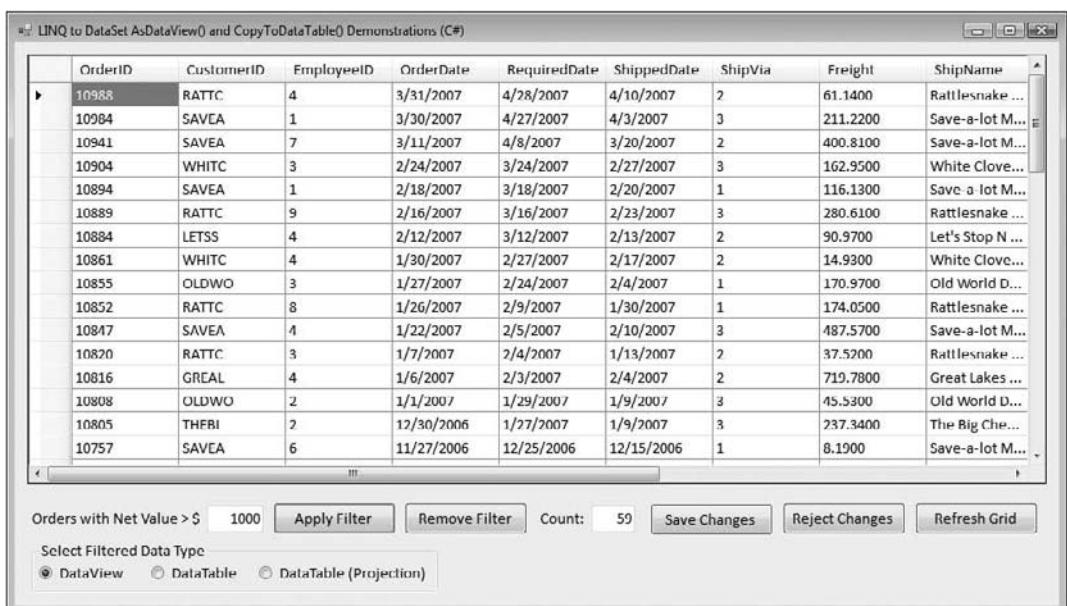


Figure 6-5

## Part III: Applying Domain-Specific LINQ Implementations

The *DataSetWindowsAppCS.sln* project has *DataTable* and *DataTable (Projection)* options that demonstrate a *CopyToDataTable()* extension method that enables generating a *DataTable* from a LINQ query projection. The next section describes using the *CopyToDataTable()* method.

The *DataSetWebAppCS* and VB (see Figure 6-6) projects emulate the *AsDataView()* feature of the Windows versions, but the *GridView* control is read-only when filtered because its *DataSource* property connects directly to the LINQ query. A *CopyToDataTable()* option would solve the read-only problem, but the *DataTable* of an *ObjectDataSource* control is read-only and can't be dynamically updated easily. If you need an updatable filtered display, you're probably better off writing a parameterized SQL SELECT statement for an *odsFilteredOrders* *ObjectDataSource* control and alternating the *GridView*'s *DataSource* property with *odsOrders*.

OrderID	CustomerID	EmployeeID	OrderDate	RequiredDate	ShippedDate	ShipVia	Freight	ShipName
Edit Select 10992	THFBI	1	4/1/2007	4/29/2007	4/3/2007	3	\$4.27	The Big Cheese
Edit Select 10988	RATTC	4	3/31/2007	4/28/2007	4/10/2007	2	\$61.14	Rattlesnake Canyon Grocery
Edit Select 10984	SAVEA	1	3/30/2007	4/27/2007	4/3/2007	3	\$211.22	Save-a-lot Markets
Edit Select 10983	SAVEA	2	3/27/2007	4/24/2007	4/6/2007	2	\$657.54	Save-a-lot Markets
Edit Select 10974	SPLIR	3	3/25/2007	4/8/2007	4/3/2007	3	\$12.96	Split Rail Beer & Ale
Edit Select 10965	OLDWO	6	3/20/2007	4/17/2007	3/30/2007	3	\$144.38	Old World Delicatessen
Edit Select 10941	SAVEA	7	3/11/2007	4/8/2007	3/20/2007	2	\$100.81	Save-a-lot Markets
Edit Select 10936	GREAL	3	3/9/2007	4/6/2007	3/18/2007	2	\$33.68	Great Lakes Food Market
Edit Select 10904	WHITC	3	2/24/2007	3/24/2007	2/27/2007	3	\$162.95	White Clover Markets
Edit Select 10894	SAVEA	1	2/18/2007	3/18/2007	2/20/2007	1	\$116.13	Save-a-lot Markets
Edit Select 10889	RATTC	9	2/16/2007	3/16/2007	2/23/2007	3	\$280.61	Rattlesnake Canyon Grocery
Edit Select 10884	LETSS	4	2/12/2007	3/12/2007	2/13/2007	2	\$90.97	Let's Stop N Shop
Edit Select 10883	LONEP	8	2/12/2007	3/12/2007	2/20/2007	3	\$0.53	Lonesome Pine Restaurant
Edit Select 10882	SAVEA	4	2/11/2007	3/11/2007	2/20/2007	3	\$23.10	Save-a-lot Markets
Edit Select 10867	LONEP	6	2/3/2007	3/17/2007	2/11/2007	1	\$1.93	Lonesome Pine Restaurant

Figure 6-6

## Copying LINQ Query Results to DataTables

Creating a new or populating an existing *DataTable* with *Expression.CopyToDataTable()* extension method is an alternative to the *Expression.AsDataView()* method. Unlike a *DataView()*, new *DataTable*s you create have no relation to existing objects by default, but you can specify inserting/updating an existing table by using one of *CopyToDataTable()*'s two overloads:

## C# 3.0

```
public static DataTable CopyToDataTable<T>
    (this IEnumerable<T> source) where T : DataRow

public static void CopyToDataTable<T>
    (this IEnumerable<T> source,
     DataTable table,
     LoadOption options) where T : DataRow

public static void CopyToDataTable<T>
    (this IEnumerable<T> source,
     DataTable table,
     LoadOption options,
     FillErrorEventHandler errorHandler) where T : DataRow
```

## VB 9.0

```
Public Shared Function CopyToDataTable(Of T As DataRow) _
(source As IEnumerable(Of T)) As DataTable

<ExtensionAttribute> _
Public Shared Sub CopyToDataTable(Of T As DataRow)
    (source As IEnumerable(Of T), _
     table As DataTable, _
     options As LoadOption)

<ExtensionAttribute> _
Public Shared Sub CopyToDataTable(Of T As DataRow)
    (source As IEnumerable(Of T), _
     table As DataTable, _
     options As LoadOption, _
     errorHandler As FillErrorEventHandler)
```

If you supply the `DataTable` as `table`, it must be empty or match the `DataRow<T>` produced by the query. The last overload lets you specify a custom error handler for exceptions when filling `DataTables`.

The following table describes the three members of the `LoadOptions` enumeration for the extension method's two overloads:

LoadOption	Description
OverwriteChanges	Values copied to this row are written to the current and original value of the data for each column.
PreserveChanges	Values copied to this row are written to the original value of each column. The current version is unchanged (default).
Upsert	Values copied to this row are written to the current version of each column. The original version is unchanged.

## Part III: Applying Domain-Specific LINQ Implementations

---

### Copying Typed DataRows

Like the `AsDataGridView()` method, the `CopyToDataTable()` extension method only accepts `IEnumerable<DataRow>` types. Neither method handles anonymous types generated by `Select()` or `SelectMany()` projections, which are the subject of the next section.

After you apply the filter expression to generate a sequence and create the data table with code like the following, you name it and add it to the `DataSet`'s `Tables` collection. Preserving updates requires adding code to create `DataAdapter` at runtime.

*Adding a DataAdapter at run time is beyond the scope of this chapter. Bill Vaughn's "Weaning Developers from the CommandBuilder" article from the MSDN Library (<http://msdn2.microsoft.com/en-us/library/ms971491.aspx>) has detailed instructions for creating a DataAdapter with VB.*

#### C# 3.0

```
private void btnApplyFilter_Click(object sender, System.EventArgs e)
{
    // ...
    var FilteredOrders = from o in dsNwindT.Orders
        where o.GetOrder_DetailsRows().Sum(d => d.Quantity *
            d.UnitPrice * (Decimal)(1 - d.Discount)) >
            Convert.ToDecimal(txtValue.Text)
        select o;

    DataTable dtOrders = FilteredOrders.CopyToDataTable();
    dtOrders.TableName = "OrdersCopy";
    dsNwindT.Tables.Add(dtOrders);
    bsOrders.DataSource = dsNwindT;
    bsOrders.DataMember = "OrdersCopy";
    bsOrders.ResetBindings(false);
}
```

#### VB 9.0

```
Private Sub btnApplyFilter_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnApplyFilter.Click
    ' ...
    Dim FilteredOrders = From o In dsNwindT.Orders
        Where o.GetOrder_DetailsRows().Sum(Function(d) _
            d.Quantity * _
            d.UnitPrice * CType((1 - d.Discount), Decimal) > _
            CType(txtValue.Text, Decimal))
        Select o;

    Dim dtOrders As DataTable = FilteredOrders.CopyToDataTable();
    dtOrders.TableName = "OrdersCopy";
    dsNwindT.Tables.Add(dtOrders);
    bsOrders.DataSource = dsNwindT;
    bsOrders.DataMember = "OrdersCopy";
    bsOrders.ResetBindings(False);
End Sub
```

*The preceding code is part of the click event handler for the Apply Filter button and represents the code that runs when you select the DataTable option.*

If your LINQ query doesn't generate an anonymous type, it's simpler to apply the `AsDataView()` rather than the `CopyToDataTable()` method.

## Processing Anonymous Types from Projections

Andrew Conrad, developer lead for Microsoft's .NET Data Services (Project Astoria), has sample code for a custom `CopyToDataTable()` extension method that accepts generic types other than `DataRow`s (<http://blogs.msdn.com/aconrad/archive/2007/09/07/science-project.aspx>). His implementation doesn't accept `Nullable<T>` types, but a reader provided a fix to support them. However, it's usually simpler to write your own code to process anonymous types than to use a patched copy of a custom extension method.

The following code, which also is part of the click event handler for the Apply Filter button, runs when you select the DataTable (Projection) option, clones `dtProj` from the `OrdersDataTable` type, populates it from a projection that omits five properties of the `Orders` object, and adds it to the `DataSet`'s `Tables` collection.

### C# 3.0

```
private void btnApplyFilter_Click(object sender, System.EventArgs e)
{
    // ...
    NorthwindTyped.OrdersDataTable dtProj =
        (NorthwindTyped.OrdersDataTable)(dsNwindT.Orders.Clone());
    // ...
    // DataTable populated from projection
    var OrdersProjection = from o in dsNwindT.Orders
                           where o.GetOrder_DetailsRows().Sum(d => d.Quantity *
                               d.UnitPrice * (Decimal)(1 - d.Discount)) >
                               Convert.ToDecimal(txtValue.Text)
                           select new
                           {
                               o.OrderID,
                               o.CustomerID,
                               o.EmployeeID,
                               o.OrderDate,
                               o.RequiredDate,
                               o.ShippedDate,
                               o.ShipVia,
                               o.Freight,
                               o.ShipCountry
                           };
    foreach (var o in OrdersProjection)
    {
        DataRow projRow = dtProj.NewRow();
        projRow[0] = o.OrderID;
        projRow[1] = o.CustomerID;
        projRow[2] = o.EmployeeID;
```

## Part III: Applying Domain-Specific LINQ Implementations

---

```
    projRow[3] = o.OrderDate;
    projRow[4] = o.RequiredDate;
    projRow[5] = o.ShippedDate;
    projRow[6] = o.ShipVia;
    projRow[7] = o.Freight;
    ' To preserve the row's presence in query
    projRow[13] = o.ShipCountry
    dtProj.Rows.Add(projRow);
}
dtProj.TableName = "OrdersProj";
dsNwindT.Tables.Add(dtProj);

bsOrders.DataSource = dsNwindT;
bsOrders.DataMember = "OrdersProj";
bsOrders.ResetBindings(false);
}
```

### VB 9.0

```
Private Sub btnApplyFilter_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnApplyFilter.Click
    ...
    Dim dtProj As DataTable = CType(dsNwindT.Orders.Clone(), _
        NorthwindTyped.OrdersDataTable)
    ...
    ' DataTable populated from projection
    Dim OrdersProjection = From o In dsNwindT.Orders _
        Where o.GetOrder_DetailsRows().Sum(Function(d) _
            d.Quantity * _
            d.UnitPrice * CType(1 - d.Discount, Decimal)) > _
            CType(txtValue.Text, Decimal) _
        Select {o.OrderID, o.CustomerID, o.EmployeeID, _
            o.OrderDate, o.RequiredDate, o.ShippedDate, _
            o.ShipVia, o.Freight, o.ShipCountry}

    For Each o In OrdersProjection
        Dim projRow As DataRow = dtProj.NewRow()
        projRow(0) = o.OrderID
        projRow(1) = o.CustomerID
        projRow(2) = o.EmployeeID
        projRow(3) = o.OrderDate
        projRow(4) = o.RequiredDate
        projRow(5) = o.ShippedDate
        projRow(6) = o.ShipVia
        projRow(7) = o.Freight
        ' To preserve the row's presence in query
        projRow(13) = o.ShipCountry
        dtProj.Rows.Add(projRow)
    Next o
    dtProj.TableName = "OrdersProj"
    dsNwindT.Tables.Add(dtProj)
```

```
        dtProj.AcceptChanges()

        bsOrders.DataSource = dsNwindT
        bsOrders.DataMember = "OrdersProj"
        bsOrders.ResetBindings(False)
End Sub
```

Saving changes is trickier because `dtProj` doesn't have the original data values to support optimistic concurrency management. The `newUpdateSql` UPDATE command doesn't include original values in its WHERE clause, so it succeeds without raising a `DBConcurrencyException` error. The `origUpdateSql` string restores the UPDATE command to its previous state with original values.

### C# 3.0

```
private void btnSaveChanges_Click(object sender, System.EventArgs e)
{
    if (dsNwindT.HasChanges())
    {
        if (dtProj != null && dtProj.GetChanges() != null)
        {
            try
            {
                // Update only the row with the manual changes
                taOrders.Adapter.UpdateCommand.CommandText = newUpdateSql;
                NorthwindTyped.OrdersDataTable projChanges =
                    (NorthwindTyped.OrdersDataTable)dtProj
                    .GetChanges(DataRowState.Modified);
                if (projChanges.Count() > 0)
                {
                    taOrders.Update(projChanges);
                    projChanges.Clear();
                }
            }
            catch (DBConcurrencyException ex)
            {
                MessageBox.Show("Concurrency exception occurred on update.");
                ex.Row.ClearErrors();
            }
            finally
            {
                taOrders.Adapter.UpdateCommand.CommandText = origUpdateSql;
            }
        }
        taOrders.Update(dsNwindT.Orders);
        dsNwindT.AcceptChanges();
    }
}
```

## Part III: Applying Domain-Specific LINQ Implementations

---

### VB 9.0

```
Private Sub btnSaveChanges_Click(ByVal sender As Object, ByVal e As System.EventArgs)
    If dsNwindT.HasChanges() Then
        If dtProj IsNot Nothing AndAlso dtProj.GetChanges() IsNot Nothing Then
            Try
                ' Update only the row with the manual changes
                taOrders.Adapter.UpdateCommand.CommandText = newUpdateSql
                Dim projChanges As NorthwindTyped.OrdersDataTable = _
                    CType(dtProj.GetChanges(DataRowState.Modified), _
                    NorthwindTyped.OrdersDataTable)
                If projChanges.Count() > 0 Then
                    taOrders.Update(projChanges)
                    projChanges.Clear()
                End If
            Catch ex As DBConcurrencyException
                MessageBox.Show("Concurrency exception occurred on update.")
                ex.Row.ClearErrors()
            Finally
                taOrders.Adapter.UpdateCommand.CommandText = origUpdateSql
            End Try
        End If
        taOrders.Update(dsNwindT.Orders)
        dsNwindT.AcceptChanges()
    End If
End Sub
```

The primary issue with this approach is substituting “last writer wins” for optimistic, value-based concurrency with changes to the filtered dataset.

## Summary

LINQ to DataSet is a specialized LINQ implementation for expanding the capabilities of typed or untyped `DataSet`s' `DataViews` and `DataTable`s. `DataSet`s don't support joining `DataTable`s or `DataViews`, so applying LINQ `Join()` and `GroupJoin()` operators overcomes that deficiency. If the `DataSet` includes 1:many `DataRelations` to child `DataTable`s, you can substitute multiple `From y In x.GetChildRows("RelationName")` statements to create hierarchically joined data sources. This chapter's `UntypedAndTypedDataSetsC#` and VB sample projects demonstrate both methods of joining `DataTable`s.

SQL-like expressions for `DataView.RowFilter` property values are limited to simple `WHERE` clauses (without `WHERE`), so another application for LINQ to `DataSet` queries is generating filtered `DataViews` or `DataTable`s by writing complex constraints and applying the `AsDataView()` or `CopyToDataTable()` method to the resulting sequence. The `DataSetsWebAppC#` and VB sample projects demonstrate that `GridView` controls connected to LINQ-filtered `DataViews` are read-only. `DataSetsWindowsAppC#` and VB show you how to create updatable filtered views for Windows apps.

Neither the `AsDataView()` or `CopyToDataTable()` methods support the anonymous types generated by LINQ queries with projections, so you must write custom code to emulate the `CopyToDataTable()` method if you must use such queries as the data source for `DataTable`s. The chapter's final code examples show you how to create an updatable, LINQ-filtered `DataTable`.

# 7

## Manipulating Documents with LINQ to XML

General-purpose computer programming languages have two components: the language itself and a set of compatible application programming interfaces (APIs) that target specific task or domain categories. As the programming tasks or domain that a particular API targets become more common, users request that the API be integrated into the language. The organizations that control the language's development usually comply with these requests because integration with the language makes use of the former API more convenient. LINQ to XML carries API integration to a new extreme by leapfrogging .NET's API-based `System.Xml` classes with an implementation that dramatically simplifies querying and generating XML Infosets and is embedded in the CLR. LINQ to XML provides XML Infoset navigation and query capabilities that equal or surpasses XQuery 1.0 and rival XPath 2.0. For document transformation, LINQ to XML can replace XQuery 1.0 or XSLT 1.0+ in the majority of common use cases.

The *LINQ to XML* topic of Chapter 1 introduced you to value-based queries against single and multiply-joined documents. This chapter extends those basic examples to queries for navigating real-world XML documents and composable code for generating hierarchical Infosets from Visual Basic 9.0's literal XML data type and the C# alternative, functional construction.

*This chapter assumes familiarity with XML entities (such as elements, attributes, names, namespaces, declarations and Infosets) and acquaintance with XML schemas but doesn't require experience with XPath, XSLT, or XQuery languages or APIs.*

### Integrating XML into the CLR

Classes in .NET's `System.Xml` namespace represents a good example of integration of XML-oriented APIs into the CLR's C# and Visual Basic, as well as IronPython, IronRuby, and other members of the new Dynamic Language Runtime (DLR). Microsoft developed the CLR in the late 1990s during the heyday of XML as the future Lingua Franca of cross-platform data integration.

## Part III: Applying Domain-Specific LINQ Implementations

---

Therefore, .NET 1.0 and later Frameworks incorporated classes that represented the XML APIs, specifications, and domain-specific languages (DSLs) listed in the following table.

XML Specification, API or DSL	.NET Framework Class
XML Document Object Model (XML DOM)	XmlDocument
XML Schema (XSD)	XSD Inference tool, XmlReader (obsoletes XmlValidatingReader)
Reading XML streams (forward-only)	XmlReader (in-memory pull model)
Writing XML streams (forward-only)	XmlWriter (in-memory)
Object to XML mapping	XmlSerializer, DataContractSerializer, NetDataContractSerializer, Atom 2.0 and RSS 3.0 feed formatters
XML Path Language (XPath)	XPathNavigator
XML Stylesheet Language/Transform (XSLT)	xslCompiledTransform, XSLT Compiler (Xsltc.exe)

XQuery 1.0 became an official W3C recommendation on January 23, 2007 after several years of development and testing. XQuery is the foundation of SQL Server 2005+'s query language for the `xml` data type. A few years ago, Microsoft operated an XQuery demonstration site ([www.xquery.com](http://www.xquery.com)) and the beta version of Visual Studio 2005 (codenamed Whidbey) included an XQuery class, which was removed before release. Microsoft's December 2004 excuse for removing the XQuery API implementation from .NET was that "neither XQuery nor XSLT are going to be W3C Recommendations in the timeframe of the Whidbey release of the .NET framework and Visual Studio." However, the SQL Server team didn't remove the XQuery tool from SQL Server 2005. In hindsight, the obvious reason was the accelerating development of the Xen (also called X#) and Cω (C-Omega) programming languages, which morphed into LINQ — and more particularly LINQ to XML — during their incubation period at Microsoft Research.

## **Minimizing XML/Object Mismatch with Xen**

Xen was the subject of the seminal "Programming with Circles, Triangles and Rectangles" presentation at the XML Conference & Exposition 2003 ([www.idealalliance.org/papers/dx\\_xml03/papers/06-02-01/06-02-01.html](http://www.idealalliance.org/papers/dx_xml03/papers/06-02-01/06-02-01.html)) by Microsoft Research's Eric Meijer and Wolfram Schulte, and Gavin Bierman, then at the University of Cambridge's Computer Laboratory and now a researcher in the Programming Principles and Tools Group at Microsoft Research in Cambridge (UK). "Circles" in the title represented objects, "triangles" stood for XML's hierarchical trees, and "rectangles" referred to relational tables. Xen introduced object/XML mapping beyond that provided by the `xsd.exe` tool for creating C# or VB classes from XML schemas. Xen also enabled programmers to use XML fragments as object literals, and defined embedded dynamic expressions (delimited with French braces). The authors later called object literals "syntactic sugar for serialized object graphs." Xen also implemented the `IEnumerable` interface, XPath-style filtering, and the `yield` method for streams of element or attribute values as part of the language. The 2003 presentation didn't include programming details for "rectangles."

### Querying XML with Cω

Cω integrated XML Infoset documents and relational tables with object models and provided “query-like capabilities” with “simple XPath-like path expressions,” as described in “The Essence of Data Access in Cω: The Power is in the Dot” presentation (<http://research.microsoft.com/Users/gmb/Papers/ecoop-corrected.pdf>) by the same authors to the 19<sup>th</sup> European Conference on Object-Oriented Programming in July 2005. Cω also introduced *anonymous structs*, which correspond to *anonymous types*, and introduced LINQ’s `from...in...select...where` query syntax in its original, `select-first` sequence, as in the following example:

#### Cω (Prototype Compiler)

```
result = select <book-with-prices>
            <title>{a.title}</title>
            <price-A>{a.price}</price-A>
            <price-BN>{bn.price}</price-BN>
        </book-with-prices>
    from book a in A.book, book bn in BN.book
    where a.title == bn.title
```

Values from data bindings replace the XML placeholders ({ . . . }), which have morphed to <%=. . . %> in LINQ to XML. Cω also introduced the first notion of an equi-join between documents in the preceding `where` clause. LINQ to XML enables joining documents to in-memory and LINQ to SQL objects, as you’ll see in the later “Performing Heterogeneous Joins and Lookup Operations” section.

## The System.Xml.Linq Namespace

The `System.Xml.Linq` namespace contains all LINQ to XML-related classes. Using LINQ to XML features requires a reference to the `System.Xml.Linq.dll` v3.5 library from the `\Program Files\Reference Assemblies\Microsoft\Framework\v3.5` folder, which is present by default in all VB 9.0 and C# 3.0 projects. Basic LINQ to XML features also are included in Silverlight 2.0.

*Silverlight 2 documentation’s “Processing XML in the .NET Framework vs. the .NET Framework for Silverlight” topic ([http://msdn.microsoft.com/en-us/library/cc189053\(VS.95\).aspx](http://msdn.microsoft.com/en-us/library/cc189053(VS.95).aspx)) explains the differences between the Silverlight 2 and .NET 3.5 SP1 LINQ to XML implementations.*

Figure 7-1 is LINQ to XML’s class hierarchy diagram, which is the same as Chapter 1’s Figure 1-12. The hierarchy is determined by the class’s inheritance, so `XDocument` and `XElement` are at the bottom of the hierarchy because they inherit from `XContainer` which inherits from `XNode` and `XObject`. The level at which the object appears in the hierarchy doesn’t indicate its importance to programming with LINQ to XML; `XElement` is the fundamental LINQ to XML object.

## Part III: Applying Domain-Specific LINQ Implementations

---

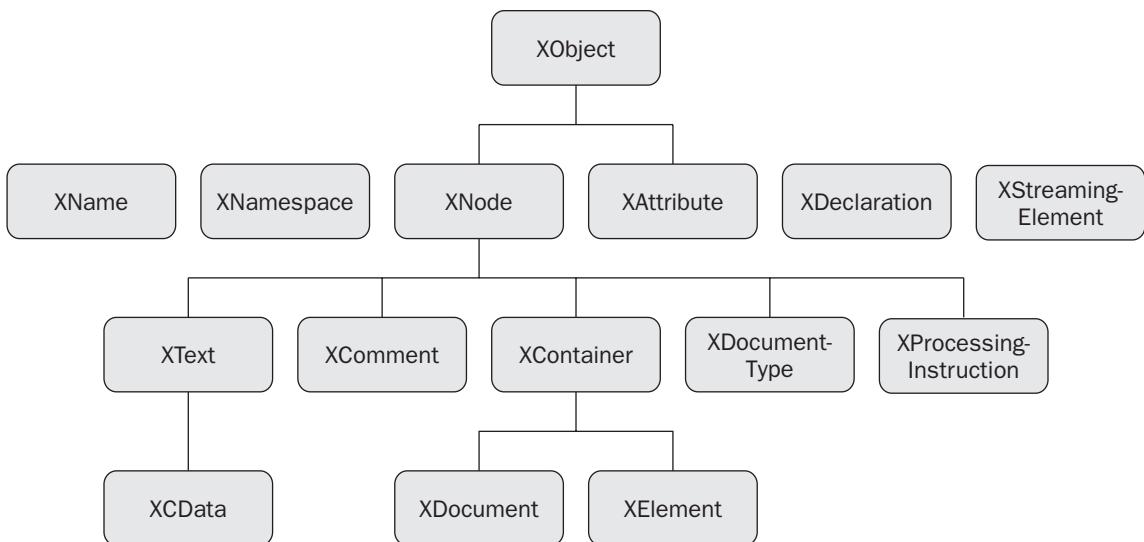


Figure 7-1

`XObject` is the abstract common base class for `XNode` and `XAttribute`. It provides annotations, which are not part of the XML Infoset, and raises an `XObjectChanged` event when an `XNode` or `XAttribute` is added to or removed from an `XContainer`, is renamed, or changes value.

Most object types in Figure 7-1 represent XML entities whose names derive from the corresponding named entities of the Extensible Markup Language (XML) 1.1 (Second Edition) W3C recommendation. Exceptions are `XContainer`, from which `XElement` and `XDocument` inherit, `XStreamingElement`, which represents streaming content, `XDocumentType`, which represents a Document Type Definition (DTD), and `XObject`, which has no corresponding entity in the recommendation.

The XML DOM requires a topmost  `XmlDocument` object to which you add  `XElement` objects. Use of LINQ to XML's  `XmlDocument` object is optional, so you can manage document fragments. You can manipulate  `XElement`s by creating them, loading them from an XML file, or saving them to a writer instance. The most use for an added  `XmlDocument` object is to include an XML declaration ( `XDeclaration`), processing instruction ( `XProcessingInstruction`), comment ( `XComment`), or any combination of these elements to the start of the document.  `XAttribute` objects, which inherit directly from  `XObject`, specify attribute name-value pairs.

In addition to the XML object classes shown in Figure 7-1, the  `Extensions` class contains the LINQ to XML-specific extension methods listed in the following table.

Extension Method	Returns Collection of or Performs Action
Ancestors	Elements that contains the ancestors of every node in the source collection
AncestorsAndSelf	Elements that contains every element in the source collection, and the ancestors of every element in the source collection
Attributes	Attributes of every element in the source collection
DescendantNodes (T)	Descendant nodes of every document and element in the source collection
DescendantNodesAndSelf	Nodes that contains every element in the source collection, and the descendant nodes of every element in the source collection
Descendants	Elements that contains the descendant elements of every element and document in the source collection
DescendantsAndSelf	Every element in the source collection, and all descendant elements for every element in the source collection
Elements	Child elements of every element and document in the source collection
InDocumentOrder (T)	Nodes that contains all nodes in the source collection, sorted in document order
Nodes (T)	Child nodes of every document and element in the source collection
Remove	Removes every attribute in the source collection from its parent element

## Querying Basic XML Infosets

Chapter 1's "Querying XML Documents" section introduced you to basic LINQ to XML query syntax with a pair of simple queries against an Orders table containing Northwind Order objects. Those queries are similar to the following examples, which return Order fragments that have USA as the destination country, in reverse chronological order:

### C# 3.0

```
private void btnOrdersqry_Click(object sender, EventArgs e)
{
    // Return an XML Infoset containing US orders in descending date order
    string fileOrders = @"\\WROX\\ADONET\\Chapter07\\Data\\Nwind\\Orders.xml";
    XDocument xdOrders = XDocument.Load(fileOrders,
                                         LoadOptions.PreserveWhitespace);

    var query = from o in xdOrders.Descendants("Order")
               where o.Element("ShipCountry").Value == "USA"
```

## Part III: Applying Domain-Specific LINQ Implementations

---

```
orderby o.Element("OrderDate").Value descending
select o;

// Create the StringBuilder for the InfoSet
sbXml.Append("<?xml version=\"1.0\" encoding=\"utf-8\" standalone=\"yes\"?>");  

sbXml.Append("<Orders>\r\n");
foreach (var o in query)
    sbXml.Append(o);
sbXml.Append("</Orders>\r\n");
DisplayXml(sbXml);
}
```

*sbXml is a form-level StringBuilder and DisplayXML(sbXml) is a method that displays the resulting InfoSet in a text box and, optionally, in Internet Explorer.*

C# 3.0 late-binds element and attribute names with *axis methods*, as illustrated by the string literals for the Order element group name and ShipCountry and OrderDate element names. Therefore, C# doesn't provide IntelliSense for XML entities.

The xdOrders.Descendants("ElementName") statement returns an `IEnumerable< XElement >` type of the child elements from an `XDocument` or `XElement` type; `xdOrders` is an `<Order>` `XDocument` in this example. `o.Element("ElementName")` returns each Element and `o.Attribute("ElementName")` returns each Attribute object of that name. The `Value` property returns the first element's or attribute's concatenated content as a string, or `null/Nothing` if the element or attribute is empty. *Concatenated content* includes text, CDATA, whitespace, and significant whitespace nodes.

*C# offers an alternative syntax to retrieve the typed content (including string) of an Element object's content: `(PrimitiveType) XElement.Element("ElementName")` or `(PrimitiveType) XElement.Attribute("ElementName")`, where `(PrimitiveType)` is the CLR equivalent of an XML schema primitive type. Be sure to specify a nullable type if elements typed as value types can be missing or empty (`minOccurs="0"`).*

`XElement.Elements()` with an empty argument returns the child element(s) of an `XElement`. The `orderby/Order By` clauses in the preceding examples evaluate XML ISO 8601-formatted dates (YYYY-MM-DD [THH:MM[:SS[.SSS...]]] [{Z|+HH:MM|-HH:MM}], as in 2008-10-10T13:30-07:00), which sort correctly as text.

VB 9.0 offers a special syntax called *XML Properties* for specifying `XElement` and `XAttribute` axes that provide IntelliSense if you include the XML document and its schema as project file references. The following "Taking Advantage of VB 9.0 Axis Properties" section describes the five axis properties; this example uses the descendant axis property.

### VB 9.0

```
Private Sub btnOrdersqry_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles btnOrdersqry.Click
    ' Return an XML InfoSet containing US orders in descending date order
    Dim fileOrders As String = "\WROX\ADONET\Chapter07\Data\Nwind\Orders.xml"
    Dim xdOrders As XDocument = XDocument.Load(fileOrders, _
        LoadOptions.PreserveWhitespace)

    Dim Orders = From o In xdOrders...<Order> _
        Where o...<ShipCountry>.Value = "USA" _
        Order By o...<OrderDate>.Value Descending _
        Select o

    ' Create the StringBuilder of the InfoSet
    sbXml.Length = 0
    sbXml.Append("<?xml version=""1.0"" encoding=""utf-8"" standalone=""yes""?>")
    sbXml.Append("<Orders>" & Constants.vbCrLf)
    For Each o In orders
        'Append Order element group
        sbXml.Append(o)
    Next o
    sbXml.Append("</Orders>" & Constants.vbCrLf)
    DisplayXml(sbXml)
End Sub
```

Simplified axis property references, capability to leverage XML schemas for early binding elements and attributes as well as enabling IntelliSense, and incorporating XML literals as a first-class data type make Visual Basic the preferred language for applications that perform heavy-duty XML query and composition operations.

LINQ to XML queries use the same structure and Standard Query Operators (SQOs) as LINQ to Objects, LINQ to SQL, and LINQ to DataSet. The primary query language differences are loose typing as the result of using `XElement` and `XAttribute` as the primary data types and `string` as the sole `Value` type. The LINQ (C#) Project Sample Query Explorer and LINQ VB Project Sample Query Explorer projects provide about 100 basic LINQ to XML queries in 12 categories. Because of the easy availability of sample query syntax for basic queries, the remainder of this chapter and its sample code concentrates on more complex queries.

*The POXQueriesCS.sln and POXQueriesVB.sln projects in the \WROX\ADONET\Chapter07\CS and . . .\VB folders contain the preceding sample query as well as a more complex query that's described in later sections.*

## **Inferring a Schema and Enabling IntelliSense for VB Queries**

As mentioned earlier, adding a reference to the XML source document and its schema (.xsd) file to your project enables IntelliSense for VB's axis property syntax. If you don't have a valid .xsd file for your source document, Visual Studio 2008's Standard Edition and higher has an XML editor that includes a schema inference engine.

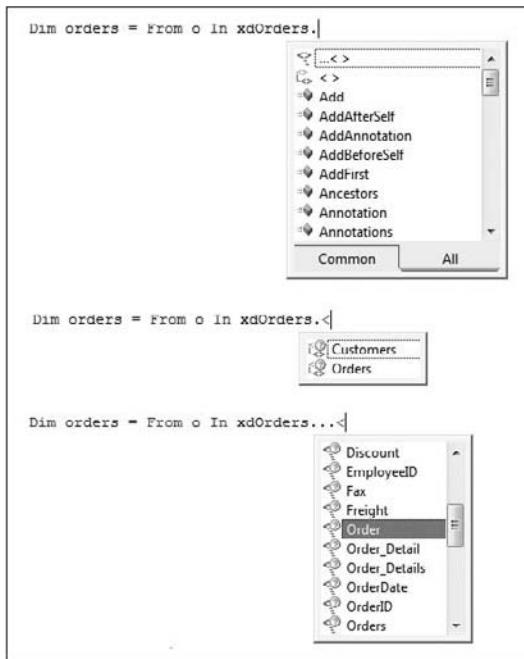
## Part III: Applying Domain-Specific LINQ Implementations

---

If you're using Visual Basic 2008 Express Edition, you can download XML to Schema Tool for Visual Basic 2008 add-in from MSDN's Visual Basic Developer site (<http://msdn2.microsoft.com/en-us/vbasic/bb840042.aspx>). The download page includes instructions for installation and use of the add-in. You must install and use this tool if your XML file uses namespace prefixes; see the later "XML Namespaces in VB 9.0" section for installation instructions.

To infer and save a schema for an XML source document in a VB project, do the following:

1. Navigate to the XML source document, right-click its node, select Open With, and choose Open in Microsoft Visual Studio 2008 to open the document in the XML editor window.
2. Optionally, click the XML Editor toolbar's Format the Whole Document button to apply standard indentation to the document, and save it in the desired location.
3. Choose XML, Create Schema to open a new *SourceXmlName.xsd* window with the completed schema.
4. Review the XML schema data types, especially for numeric values, to verify that they correspond to your desired CLR data type. The inference engine chooses the minimum width type that accommodates the maximum numerical value in the document. Many changes are required to the schema for the \Wrox\ADONET\Chapter07\Data\Nwind\Orders.xml source document.
5. Review `minOccurs` attribute values for `Strings` and value types. Specify `minOccurs="0"` to enable null `Strings` and `Nullable(Of T)` value types, such as `ShippedDate` for the Northwind orders.
6. Add Imports directives for each namespace in your XML document, as described in the later "Multiple Namespaces in XML Literal Queries" section.
7. Test IntelliSense by creating a sample `XDocument` from a file and then starting a LINQ to XML query with a statement fragment such as `Dim Orders = From o In xdOrders.` (include the dot) to display the `XDocument`'s properties and methods in the IntelliSense list (see Figure 7-2, top).
8. Type an opening angle bracket to display the `XDocument`'s accessible child elements, `Customers` and `Orders` in the IntelliSense list (see Figure 7-2, center).
9. Backspace and type two additional dots plus an opening angle bracket to display all accessible axis property references in the IntelliSense list (see Figure 7-2, bottom).



**Figure 7-2**

If you share the schema with multiple projects, you can create a link to the schema by opening the Add Existing Item dialog's Add button's drop-down list and choosing Add as Link.

## Taking Advantage of VB 9.0 Axis Properties

The following table lists VB 9.0's symbolic syntax for axis property references with the symbol for each reference's use and a description of its operation.

Axis Property Reference	Example	Description
Child axis	Order.<CustomerID>	Gets all CustomerID child elements of the Order element group
Attribute axis	Order.@OrderID	Gets all OrderID attributes of the Order element group
Descendant axis	Customer...<ProductID>	Gets all ProductID elements of the Customer element group, regardless of how low in the hierarchy they occur
Extension indexer	Customer...<ProductID>(0)	Gets the first ProductID element from the Customer element group
Value	Customer...<ProductID>.Value	Gets the string representation of the first content in the ProductID sequence, or Nothing for an empty sequence

## Part III: Applying Domain-Specific LINQ Implementations

---

Even if you don't enable IntelliSense by adding an XML schema to your project, VB's symbolic axis property references save a substantial amount of typing.

Compare the C# and VB versions of the following queries from the POXQueriesCS.sln and POXQueriesVB.sln projects that return from the Orders.xml InfoSet an anonymous type with aggregate values for net order amounts and total line item counts by country:

### C# 3.0

```
var Orders3 = from o in xdOrders.Descendants("Order")
              from d in o.Descendants("Order_Detail")
              orderby (string)o.Element("ShipCountry")
              group d by (string)o.Element("ShipCountry") into g
              select new { Country = g.Key,
                          LineItems = g.Count(),
                          Subtotal = g.Sum(d => (Decimal)d.Element("Quantity") *
                                         (Decimal)d.Element("UnitPrice") *
                                         (1 - (Decimal)d.Element("Discount")) )};
```

### VB 9.0

```
Dim Orders3 = From o In xdOrders...<Order>, _
                 d In o...<Order_Details> _
                 Order By o.<ShipCountry>.Value _
                 Group d By Country = o.<ShipCountry>.Value _
                 Into Subtotal = Sum(CDec(d...<Quantity>.Value) * _
                                     CDec(d...<UnitPrice>.Value) * _
                                     (1 - CDec(d...<Discount>.Value))), _
                 LineItems = Count()
```

C# is terser than VB in most cases, but the preceding C# query is 329 characters without spaces while its VB equivalent is 252 characters; the C# query is 30% more verbose. Notice that the VB version has a simple query structure that doesn't need a lambda function or a Select projection.

*Many of this chapter's examples use grouping and aggregation because there are very few sample LINQ to XML queries of this type in VS 2008's online help, Microsoft whitepapers, or other LINQ to XML documentation such as books and blogs. The two Query Explorer projects offer only three simple groupby/Group By samples that don't use aggregates.*

*Neither the C# nor the VB version of the second Group Orders by Customer example (XLinq55) from the CSsharpSamples.zip and VBSamples.zip files' release version (date 11/8/2007) work as expected due to misuse of the Value property. The original and working versions of these samples are the GroupOrdersByCustomerCS.sln and GroupOrdersByCustomerVB.sln console projects in this chapter's sample folders.*

## **Implicit versus Explicit Typing of Element and Attribute Content**

As mentioned in the preceding table and note, the `Value` property returns the content of an `XElement` as `String.Empty/String`.`Empty` if the element is empty or missing. You can set an `XElement`'s content by assigning a string to the `Value` property. Adding an XML schema to the project doesn't strongly type LINQ to XML's element values, so the element values are said to be *implicitly typed*.

## Chapter 7: Manipulating Documents with LINQ to XML

---

Following are the C# and VB implementations of the `Value` method based on disassembly with Red Gate's (formerly Lutz Roeder's) Reflector application:

### C# 3.0

```
public string Value
{
    get
    {
        if (base.content == null)
        {
            return string.Empty;
        }
        string content = base.content as string;
        if (content != null)
        {
            return content;
        }
        StringBuilder sb = new StringBuilder();
        this.AppendText(sb);
        return sb.ToString();
    }
    set
    {
        if (value == null)
        {
            throw new ArgumentNullException("value");
        }
        base.RemoveNodes();
        base.Add(value);
    }
}
```

### VB 9.0

```
Public Property Value As String
    Get
        If ( MyBase.content Is Nothing) Then
            Return String.Empty
        End If
        Dim content As String = TryCast(MyBase.content, String)
        If (Not content Is Nothing) Then
            Return content
        End If
        Dim sb As New StringBuilder
        Me.AppendText(sb)
        Return sb.ToString
    End Get
    Set(ByVal value As String)
        If (value Is Nothing) Then
            Throw New ArgumentNullException("value")
        End If
        MyBase.RemoveNodes
        MyBase.Add(value)
    End Set
End Property
```

## Part III: Applying Domain-Specific LINQ Implementations

---

*Using LINQ to XSD in conjunction with your LINQ to XML query strongly types the element values. LINQ to XSD is one of the subjects of Chapter 8.*

The XElement class has 25 read-only explicit operator (`op_explicit(XElement):DataType`) methods to cast an XElement's value to String or one of 24 value types, half of which are Nullable<T> types. The preceding query snippets illustrate the C# and VB syntax for explicitly typing the Quantity, UnitPrice and Discount elements as Decimal, as in `(Decimal)d.Element("UnitPrice")` for C#. You must use explicit typing when applying mathematic or aggregate operators to element values.

To explicitly type C# values to Nullable<T> types, append a ? to the data type as in `(Decimal?)d.Element("UnitPrice").Change` VB's CTypeAbbr(d...<ElementName>.Value), as in `CDec(d...<UnitPrice>.Value)`, to `CType(d...<ElementName>.Value, Nullable(Of DataType))`, as in `CType(d...<UnitPrice>.Value, Nullable(Of Decimal))`.

Following are the C# and VB implementations of the `op_explicit(XElement):DataType` method:

### C# 3.0

```
[CLSCompliant(false)]
public static explicit operator decimal?(XElement element)
{
    if (element == null)
    {
        return null;
    }
    return new decimal?(XmlConvert.ToDecimal(element.Value));
}
```

### VB 9.0

```
<CLSCompliant(False)> _
Public Shared Narrowing Operator CType(ByVal element As XElement) As Decimal?
    If (element Is Nothing) Then
        Return Nothing
    End If
    Return New Decimal?(XmlConvert.ToDecimal(element.Value))
End Function
```

XAttribute objects have a similar Value property and set of 25 `op_explicit(XAttribute):DataType` methods that use the same explicit type syntax.

## Composing XML Infosets

The traditional approach to programmatically create XML Infosets is to do the following with System.Xml's XmlDocument object, which implements the XML DOM API in .NET:

1. Create a new named XmlDocument object.
2. Create new named XElement objects.

3. Add content to `XmlElement`s as `InnerText` property values.
4. Append `XmlAttribute`s to `XmlElement`s with the `SetAttribute()` method.
5. Append `XmlElement`s to the  `XmlDocument` or other `XmlElement`s with the `AppendChild(ElementName)` method.

The preceding clearly is a very tedious, imperative process and isn't well suited to creating documents with dynamic content from databases and other data sources.

.NET's `XmlTextWriter` class offers a more straightforward approach to authoring well-formed XML documents with dynamic content. Following are examples of C# 1.0 methods and VB 7.0 procedures for .NET 1.1+ to create and save the XML representation of a Northwind `Order_Detail` object with an `XmlTextWriter`:

### C# 1.0

```
private void Order_Detail(XmlWriter xwDetail,
                         int OrderID,
                         int ProductID,
                         short Quantity,
                         decimal UnitPrice,
                         float Discount)
{
    xwDetail.WriteStartElement("Order_Detail");
    xwDetail.WriteAttributeString("OrderID", XmlConvert.ToString(OrderID));
    xwDetail.WriteElementString("ProductID", XmlConvert.ToString(ProductID));
    xwDetail.WriteElementString("Quantity", XmlConvert.ToString(Quantity));
    xwDetail.WriteElementString("UnitPrice", XmlConvert.ToString(UnitPrice));
    xwDetail.WriteElementString("Discount", XmlConvert.ToString(Discount));
    xwDetail.WriteEndElement();
}
private void XmlLiteralConstruction_Load(object sender, System.EventArgs e)
{
    string filePath =
        "\\\\WROX\\\\ADONET\\\\Chapter07\\\\Data\\\\TextAndXmlFiles\\\\Order_Detail.xml";

    // Save the XML document to a file with an XML declaration and comment
    XmlTextWriter xtwDetail = new XmlTextWriter(filePath, Encoding.UTF8);
    xtwDetail.Formatting = Formatting.Indented;
    xtwDetail.WriteStartDocument(true);
    xtwDetail.WriteComment("XmlTextWriter example");
    Order_Detail(xtwDetail, 10232, 15, 12, 5.5M, 0.05F);
    xtwDetail.Close();
}
```

## Part III: Applying Domain-Specific LINQ Implementations

---

### VB 7.0

```
Private Sub Order_Detail(ByVal xwDetail As XmlWriter, _
    ByVal OrderID As Integer, _
    ByVal ProductID As Integer, _
    ByVal Quantity As Short, _
    ByVal UnitPrice As Decimal, _
    ByVal Discount As Single)
    With xwDetail
        .WriteStartElement("Order_Detail")
        .WriteAttributeString("OrderID", XmlConvert.ToString(OrderID))
        .WriteElementString("ProductID", XmlConvert.ToString(ProductID))
        .WriteElementString("Quantity", XmlConvert.ToString(Quantity))
        .WriteElementString("UnitPrice", XmlConvert.ToString(UnitPrice))
        .WriteElementString("Discount", XmlConvert.ToString(Discount))
        .WriteEndElement()
    End With
End Sub

Private Sub XmlLiteralConstruction_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
    ' Save the XML document to a file with an XML declaration and comment
    Dim filePath As String = _
        "\WROX\ADONET\Chapter07\Data\TextAndXmlFiles\Order_Detail.xml"
    Dim xtwDetail As New XmlTextWriter(filePath, Encoding.UTF8)
    With xtwDetail
        .Formatting = Formatting.Indented
        .WriteStartDocument(True)
        .WriteComment("XmlTextWriter example")
        Order_Detail(xtwDetail, 10232, 15, 12, 5.5D, 0.05)
        .Close()
    End With
End Sub
```

Composing XML Infosets with LINQ to XML is substantially easier and more straightforward than using the XML DOM. It's also simpler as well as more intuitive than writing `XmlTextWriter` code. As is the case for queries, C# 3.0 and VB 9.0 have radically different syntax and methodology for composing XML Infosets with LINQ to XML, as you'll see in the two following sections.

*The preceding code snippets are included in the \WROX\ADONET\Chapter07\CS\FunctionalConstructionCS.sln and \WROX\ADONET\Chapter07\VB\XmlLiteralsConstructionVB.sln sample projects for the following two sections.*

## Using Functional Construction with C# 3.0

C# 3.0 uses *functional construction* to create `XDocument`, `XElement` and `XAttribute` entities declaratively. An optional `XDeclaration` argument starts an `XDocument`; otherwise, an Infoset can consist of `XElements` and optional `XAttributes` only. Functional construction relies on one of the following

actions taken by the overloaded `XElement` constructor when it's passed one the following specified argument types:

- An `XElement` object creates a child element.
- A param array of type `Object` creates an element that has complex content.
- An `XAttribute` object creates an attribute of the element.
- An object that implements `IEnumerable<T>` results in enumeration of the object's collection and addition of all objects. `XElement` or `XAttribute` objects add individually, which lets you pass the results of a LINQ query to the constructor.
- An `XText` object adds its text and changes the element to mixed content.
- Any other object type that can be converted to a string adds the element's text content.

Functional construction offers top-down, declarative document design. The following snippet from the `FunctionalConstructionCS.sln` project illustrates functional construction of an XML InfoSet with an XML Declaration, comment, and content from a tab-separated worksheet exported in default format by Excel 2007:

### C# 3.0

```
private void btnCustomers_Click(object sender, EventArgs e)
{
    string filePath = @"\\WROX\\ADONET\\Chapter07\\Data\\TextAndXmlFiles\\";
    var custsDoc = new XDocument(
        new XDeclaration("1.0", "utf-8", "yes"),
        new XComment("From Excel tab-separated-values file"),
        new XElement("Customers",
            from row in File.ReadAllLines(filePath + "Customers.txt")
            let col = row.Split('\\t')
            // Don't process first row's column names
            where !row.Contains("CompanyName")
            select new XElement("Customer",
                new XAttribute("CustomerID", col[0]),
                new XElement("CompanyName", col[1]),
                new XElement("ContactInfo",
                    new XElement("ContactName", col[2]),
                    new XElement("ContactTitle", col[3]),
                    new XElement("WorkPhone", "Missing"),
                    new XElement("MobilePhone", "Missing"),
                    new XElement("ContactEmail", "Missing")
                ),
                new XElement("BillingAddress",
                    new XElement("Address", col[4]),
                    new XElement("City", col[5]),
                    new XElement("Region", col[6]),
                    new XElement("PostalCode", col[7]),
                    new XElement("Country", col[8])
                ),
                new XElement("Communications",

```

## Part III: Applying Domain-Specific LINQ Implementations

```
        new XElement("Phone", col[9]),
        new XElement("Fax", col[10]),
        new XElement("CustomerEmail", "Missing")
    ))
};

if (chkIE.Checked)
{
    // Display the file in IE and the text box (with XML declaration)
    custsDoc.Save(filePath + "Customers.xml");
    txtXML.Text = File.ReadAllText(filePath + "Customers.xml");
    Process.Start(filePath + "Customers.xml");
}
else
{
    // No XML declaration appears unless the document is saved to a file
    txtXML.Text = custsDoc.ToString();
}
```

Figure 7-3 shows the FunctionalConstructionCS.sln project's Windows form displaying the first Customer element of the XML Infoset generated by the preceding code.

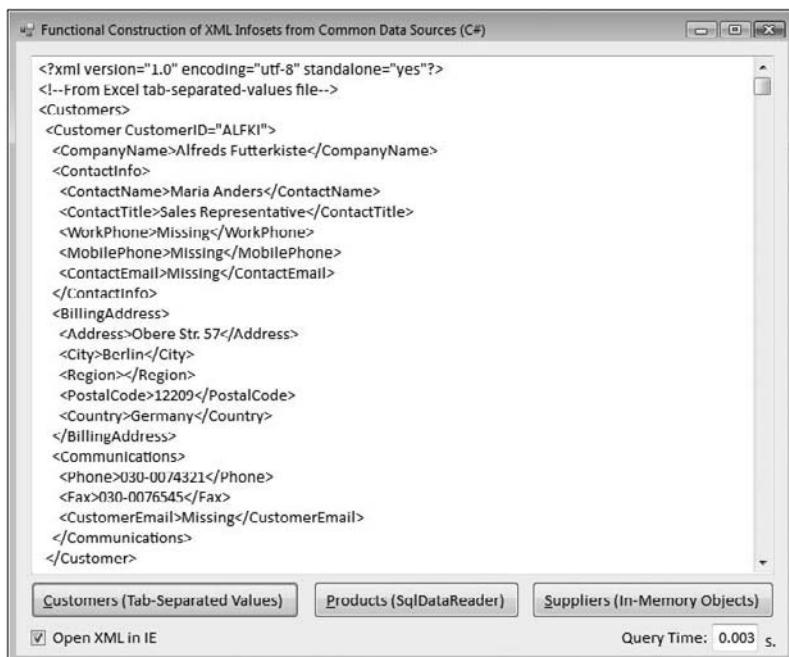


Figure 7-3

## Chapter 7: Manipulating Documents with LINQ to XML

Alternatively, you can use the `XElement.Add()` method to insert `XElements` or grouped `XElements` as siblings or children of existing elements. The following example's highlighted loop uses an `SqlDataReader` to add a `Product` group with autogenerated child `XElements` to a `Products` `XDocument` for each `Products` table row:

### C# 3.0

```
private void btnProducts_Click(object sender, EventArgs e)
{
    string strNwind = Properties.Settings.Default.NorthwindConnectionString;
    XDocument prodsDoc = null;
    SqlConnection connNwind = new SqlConnection(strNwind);
    using (SqlCommand cmdNwind = new SqlCommand("SELECT * FROM Products",
        connNwind))
    {
        connNwind.Open();
        SqlDataReader sdrNwind =
            cmdNwind.ExecuteReader(CommandBehavior.SequentialAccess);
        prodsDoc = new XDocument(
            new XDeclaration("1.0", "utf-8", "yes"),
            new XComment("From SqlDataReader over Northwind.Products"),
            new XElement("Products"));

        while (sdrNwind.Read())
        {
            var prodsElement = new XElement("Product",
                from col in Enumerable.Range(0, sdrNwind.FieldCount)
                select new XElement(sdrNwind.GetName(col), sdrNwind.GetValue(col)));
            prodsDoc.Element("Products").Add(prodsElement);
        }
        sdrNwind.Close();
    }
    // Display code is similar to the preceding example.
}
```

The preceding approach assumes that child element names are the same as column names, which is a common practice.

The third common data source from which to construct XML Infosets is an in-memory generic list or object graph. The following example uses a `List<Supplier>` to construct a `Suppliers.xml` document:

### C# 3.0

```
private void btnSuppliers_Click(object sender, EventArgs e)
{
    vendorList = CreateSupplierList();
    var vendsDoc = new XDocument(
        new XDeclaration("1.0", "utf-8", "yes"),
        new XComment("From in-memory List<Supplier>"),
        new XElement("Suppliers",
            from v in vendorList
            select new XElement("Supplier",
                new XAttribute("SupplierID", v.SupplierID),
```

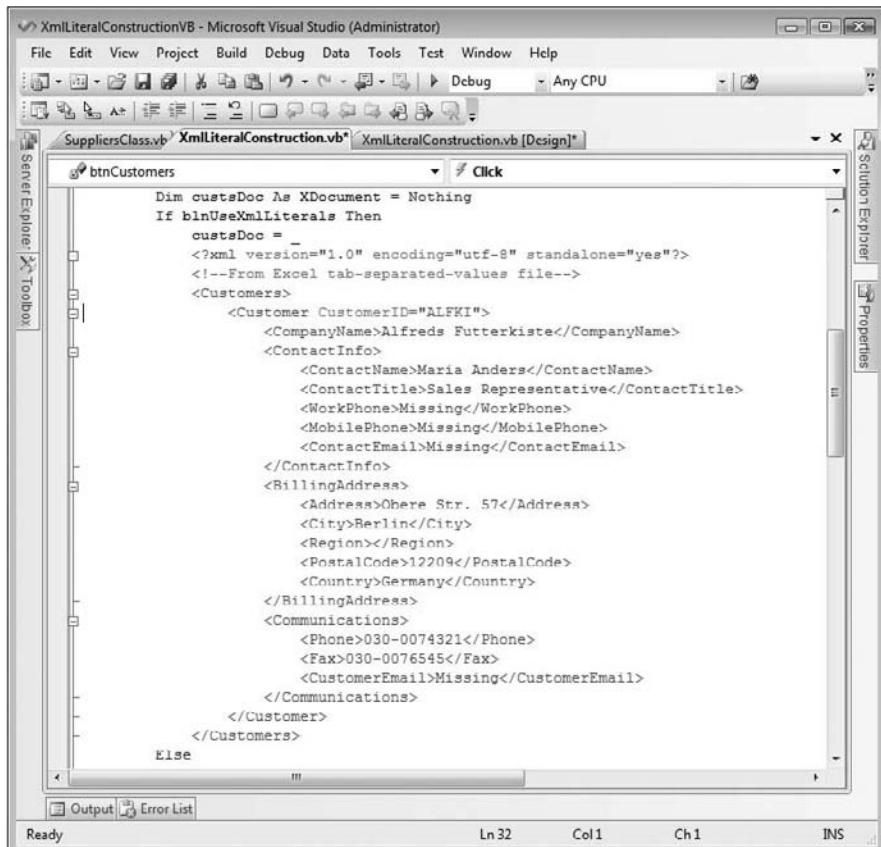
## Part III: Applying Domain-Specific LINQ Implementations

```
        new XElement("CompanyName", v.CompanyName),
        new XElement("ContactInfo",
            new XElement("ContactName", v.ContactName),
            new XElement("ContactTitle", v.ContactTitle),
            new XElement("WorkPhone", "Missing"),
            new XElement("MobilePhone", "Missing"),
            new XElement("ContactEmail", "Missing")
        ),
        new XElement("SalesAddress",
            new XElement("Address", v.Address),
            new XElement("City", v.City),
            new XElement("Region", v.Region),
            new XElement("PostalCode", v.PostalCode),
            new XElement("Country", v.Country)
        ),
        new XElement("Communications",
            new XElement("Phone", v.Phone),
            new XElement("Fax", v.Fax),
            new XElement("SupplierEmail", "Missing")
        )))
    );
    // Display code is similar to the other examples.
}
```

The VB 9.0 compiler handles functional construction also, but literal XML is the preferred coding style for most operations. The only significant changes required to port the preceding example to VB are: Change void to Function and var to Dim, add line continuation characters, remove the semicolon, and change the comment symbol. The next section has an example of VB functional construction.

## Using Literal XML Construction with VB 9.0

VB 9.0 can use functional construction to compose XML Infosets but it offers an elegant alternative: A built-in literal XML data type to compose XML documents from a string representation of the desired document structure. Literal construction lets you paste a sample XML Infoset into the VB editor and check the initial result before adding the LINQ query or assigning variables to provide element and attribute content. Figure 7-4 shows the first Customer element with the root Customers node's XDeclaration and XComment objects copied (refer to Figure 7-3) and pasted into the XmlLiteralsConstructionVB.sln project's code.



**Figure 7-4**

*When you complete the opening element tag, the VB Editor adds a corresponding closing tag.*

*You must manually add the closing root element, </Customers> for this copy and paste example.*

*Notice that line continuation characters aren't required (or allowed) for content that represents the document's structure. However, multi-line embedded VB expressions require line continuation characters.*

Embedded expressions for LINQ to XML queries and computed values for element or attribute content must be enclosed within nested <%= / => pairs, originally called expression holes, which are the same as the symbols ASP.NET uses to enclose expressions embedded in XHTML content.

*When you type the <%= characters, the VB Editor adds a space and completes the pair with => characters.*

## Part III: Applying Domain-Specific LINQ Implementations

---

The following code, which emphasizes symbol pairs, returns the same document as its C# functional construction counterpart of the preceding section:

### VB 9.0

```
Private Sub btnCustomers_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles btnCustomers.Click  
    Dim custsDoc As XDocument = _  
        <?xml version="1.0" encoding="utf-8" standalone="yes"?>  
        <!--From Excel tab-separated-values file-->  
        <Customers>  
            <%= From Row In File.ReadAllLines(filePath & "Customers.txt") _  
                Let Col = Row.Split(ControlChars.Tab) _  
                Where (Not Row.Contains("CompanyName")) _  
                Select New XElement( _  
                    <Customer CustomerID=<%= Col(0) %>>  
                    <CompanyName><%= Col(1) %></CompanyName>  
                    <ContactInfo>  
                        <ContactName><%= Col(2) %></ContactName>  
                        <ContactTitle><%= Col(3) %></ContactTitle>  
                        <WorkPhone>Missing</WorkPhone>  
                        <MobilePhone>Missing</MobilePhone>  
                        <ContactEmail>Missing</ContactEmail>  
                    </ContactInfo>  
                    <BillingAddress>  
                        <Address><%= Col(4) %></Address>  
                        <City><%= Col(5) %></City>  
                        <Region><%= Col(6) %></Region>  
                        <PostalCode><%= Col(7) %></PostalCode>  
                        <Country><%= Col(8) %></Country>  
                    </BillingAddress>  
                    <Communications>  
                        <Phone><%= Col(9) %></Phone>  
                        <Fax><%= Col(10) %></Fax>  
                        <CustomerEmail>Missing</CustomerEmail>  
                    </Communications>  
                </Customer>) _  
            %>  
        </Customers>  
        ' Display code is similar to previous examples  
End Sub
```

*The sample project btnCustomers\_Click event handler includes a port of the C# functional construction version, the singleton Customer element shown in Figure 7-4, and the preceding XML literal procedure.*

## Chapter 7: Manipulating Documents with LINQ to XML

XML literal composition for the Products InfoSet is much more complex than functional composition, so the latter technique is used in the following procedure:

### VB 9.0

```
Private Sub btnProducts_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnProducts.Click
    swTimer.Start()
    Dim strNwind As String = My.Settings.NorthwindConnectionString
    Dim prodsDoc As XDocument = Nothing
    Dim connNwind As SqlConnection = New SqlConnection(strNwind)
    Using cmdNwind As SqlCommand = _
        New SqlCommand("SELECT * FROM Products", connNwind)
        connNwind.Open()
        Dim sdrNwind As SqlDataReader = _
            cmdNwind.ExecuteReader(CommandBehavior.SequentialAccess)
        prodsDoc = New XDocument(New XDeclaration("1.0", "utf-8", "yes"), _
            New XComment("From SqlDataReader over Northwind.Products"), _
            New XElement("Products"))

        Do While sdrNwind.Read()
            Dim prodsElement = New XElement("Product", _
                From col In Enumerable.Range(0, _
                    sdrNwind.FieldCount) _
                Select New XElement(sdrNwind.GetName(col), _
                    sdrNwind.GetValue(col)))
            prodsDoc.Element("Products").Add(prodsElement)
        Loop
        sdrNwind.Close()
    End Using
    ' Display code is similar to previous examples
End Sub
```

The Suppliers.xml document's XML literal code is similar to that for the Customers.xml document:

### VB 9.0

```
Private Sub btnSuppliers_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnSuppliers.Click
    VendorsList = CreateSupplierList()
    Dim VendsDoc As XDocument = _
        <?xml version="1.0" encoding="utf-8" standalone="yes"?>
        <!--From in-memory List<Supplier>-->
        <Suppliers>
            <%= From v In VendorsList _ 
            Select New XElement( _
                <Supplier SupplierID=<%= v.SupplierID %>>
                <CompanyName><%= v.CompanyName %></CompanyName>
                <ContactInfo>
                    <ContactName><%= v.ContactName %></ContactName>
                    <ContactTitle><%= v.ContactTitle %></ContactTitle>
                    <WorkPhone>Missing</WorkPhone>
                    <MobilePhone>Missing</MobilePhone>
```

```
<ContactEmail>Missing</ContactEmail>
</ContactInfo>
<SalesAddress>
    <Address><%= v.Address %></Address>
    <City><%= v.City %></City>
    <Region><%= v.Region %></Region>
    <PostalCode><%= v.PostalCode %></PostalCode>
    <Country><%= v.Country %></Country>
</SalesAddress>
<Communications>
    <Phone><%= v.Phone %></Phone>
    <Fax><%= v.Fax %></Fax>
    <SupplierEmail>Missing</SupplierEmail>
</Communications>
</Supplier>) _
%>
</Suppliers>
' Display code is similar to previous examples
End Sub
```

# Grouping Elements and Aggregating Numeric Values of Business Documents

One of the most common applications for XPath, XSLT, and XQuery is aggregating numeric values for document-internal applications, such as totaling a single business document (for example, a sales order or invoice) and creating running totals for groups of business documents. XPath 1.0 supports `sum()` and `count()`, and XQuery 1.0, XPath 2.0, and XSLT 2.0 support `sum()`, `count()`, `avg()`, `min()` and `max()`, which have corresponding SQOs..NET 3.5 and VS 2008 SP1 don't support XQuery 1.0, XPath 2.0 or XSLT 2.0, so LINQ to XML probably is your best bet for an all-Microsoft solution that requires one of the three added aggregate functions.

*Another reason for substituting LINQ to XML for XPath, XSLT, and XQuery is to leverage the fluency you gain from LINQ to Objects, LINQ to SQL, LINQ to DataSet, or LINQ to Entities experience rather than learning a single-purpose XML domain-specific language from scratch.*

XML Infosets that represent business documents are by nature hierarchical, which aids in creating running totals for groups of documents. The following three sections demonstrate how to create hierarchical Infosets with VB's Group Join expression as well as from object graphs with many:1 and 1:many associations. The three procedures handle the `GroupJoin` (Unformatted), `Associations (Formatted)`, and `Associations (Aggregates)` buttons' Click event in the `GroupingAndAggregatingXmlVB.sln` project. The project is in your `\WROX\ADONET\Chapter07\VB\GroupingAndAggregatingXmlVB` folder. This project's data sources are classes and `List<T>` collections created by the LINQ In-Memory Object Generator (LIMOG) utility described in Chapter 4.

*There's no C# version of the `GroupingAndAggregatingXmlVB.sln` project because duplicating VB's XML literal code with functional construction is an unrewarding exercise.*

## Using GroupJoin to Produce Hierarchical Documents

If you're dealing with related source objects that don't have 1:many associations or the equivalent, the Group Join expression is your best bet for creating a hierarchical Infoset for later addition of code to produce localized and running totals. The following XML literal code generates a document with `Customers`, `Customer`, `Orders`, `Order`, `Order_Details`, and `Order_Detail` elements. Primary key values that aren't significant to customers appear as attributes and the Group Join expressions are highlighted:

### VB 9.0

```
Private Sub btnGroupJoin_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnGroupJoin.Click
    ' Create a simple hierarchical Orders and Order Details
    ' Infoset by Customer grouping
    Dim CustomerOrders = _
        <Customers>
            <%= From c In CustomerList _
                Group Join o In OrderList _
                    On o.CustomerID Equals c.CustomerID Into ord = Group _
                    Select _
                        <Customer CustomerID=<%= c.CustomerID %>>
                            <CompanyName><%= c.CompanyName %></CompanyName>
                            <City><%= c.City %></City>
                            <Country><%= c.Country %></Country>
                            <Orders>
                                <%= From o In ord _
                                    Group Join d In Order_DetailList _
                                        On d.OrderID Equals o.OrderID Into dtl = Group _
                                        Select _
                                            <Order CustomerID=<%= o.CustomerID %>>
                                                <OrderID><%= o.OrderID %></OrderID>
                                                <EmployeeID><%= o.EmployeeID %></EmployeeID>
                                                <OrderDate><%= o.OrderDate %></OrderDate>
                                                <RequiredDate><%= o.RequiredDate %></RequiredDate>
                                                <ShippedDate><%= o.ShippedDate %></ShippedDate>
                                                <ShipVia><%= o.ShipVia %></ShipVia>
                                                <Freight><%= o.Freight.Value %></Freight>
                                                <ShipName><%= o.ShipName %></ShipName>
                                                <ShipCountry><%= o.ShipCountry %></ShipCountry>
                                                <Details>
                                                    <%= From d In dtl _
                                                        Select _
                                                            <Order_Detail OrderID=<%= d.OrderID %>>
                                                                <ProductID><%= d.ProductID %></ProductID>
                                                                <Quantity><%= d.Quantity %></Quantity>
                                                                <UnitPrice><%= d.UnitPrice %></UnitPrice>
                                                                <Discount><%= d.Discount %></Discount>
                                                                </Order_Detail> %
                                                        </Details>
                                                    </Order> %
                                                </Orders>
                                            </Customer> %
                                        </Customers>
    End Sub
```

## Part III: Applying Domain-Specific LINQ Implementations

---

*Most address fields are omitted from the class definitions and object initializers for brevity.*

### Taking Advantage of 1:Many and Many:1 Associations

Using object graphs with 1:many (`EntityRef`) and associations as data sources simplifies expressions by traversing 1:many associations instead of performing a `Group Join` to establish the document hierarchy. Many:1 (`EntitySet`) associations let you substitute meaningful names or descriptions for numeric foreign-key values. The following XML literal code adds highlighted `SalesPerson`, `ShipperName`, `ProductName`, and `SKU` elements from many:1 associations and calculates an Extended line item value:

#### VB 9.0

```
Private Sub btnAssns_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnAssns.Click
    ' Create hierarchical Orders and Order Details by
    ' Customer -> Orders -> Order_Details associations
    ' Use EntityRef associations to substitute for numeric foreign key values
    Dim CustomerOrders = _
    <Customers>
    <%= From c In CustomerList _
        Select _
        <Customer>
        <CompanyName CustomerID=<%= c.CustomerID %>>
            <%= c.CompanyName %></CompanyName>
        <City><%= c.City %></City>
        <Country><%= c.Country %></Country>
    <Orders>
    <%= From o In c.Orders _
        Order By o.OrderID Descending _
        Select _
        <Order CustomerID=<%= o.CustomerID %>>
            <OrderID><%= o.OrderID %></OrderID>
            <SalesPerson EmployeeID=<%= o.EmployeeID %>>
                <%= o.Employee.FirstName & " " & o.Employee.LastName %>
            </SalesPerson>
            <OrderDate><%= o.OrderDate.Value.ToShortDateString() %></OrderDate>
            <RequiredDate>
                <%= o.RequiredDate.Value.ToShortDateString() %>
            </RequiredDate>
            <ShippedDate>
                <%= o.ShippedDate.Value.ToShortDateString() %>
            </ShippedDate>
            <ShipperName ShipperID=<%= o.ShipVia %>>
                <%= o.Shipper.CompanyName %>
            </ShipperName>
            <ShipToName><%= o.ShipName %></ShipToName>
            <ShipToCountry><%= o.ShipCountry %></ShipToCountry>
            <Details>
                <%= From d In o.Order_Details _
                    Let s = CDec(d.Quantity) * CDec(d.UnitPrice) * _
                        (1 - CDec(d.Discount)) _
                    Select _
                    <Order_Detail OrderID=<%= d.OrderID %>>
            </Details>
        </Order>
    </Orders>
</Customer>
    </Customers>
```

## Chapter 7: Manipulating Documents with LINQ to XML

```
<Quantity><%= d.Quantity %></Quantity>
<ProductID><%= d.ProductID %></ProductID>
<ProductName><%= d.Product.ProductName %></ProductName>
<SKU><%= d.Product.QuantityPerUnit %></SKU>
<UnitPrice> _
    <%= String.Format("{0:c}", d.UnitPrice) %>
</UnitPrice>
<Discount>
    <%= String.Format("{0:p1}", d.Discount) %>
</Discount>
<Extended><%= String.Format("{0:c}", s) %></Extended>
</Order_Detail> %
</Details>
</Order> %
</Orders>
</Customer> %
</Customers>
End Sub
```

Figure 7-5 shows IE 7 displaying orders for the first member of the CustomerList data source.



Figure 7-5

### **Aggregating Order\_Details and Orders Subtotals per Customer**

Code to add running totals to individual and groups of documents isn't as intuitive as literal XML and expressions for supplying calculated element and attribute content. This might explain why so few examples of LINQ to XML code with running sums are available from books, online help files, and the Internet. In most cases, a running total for each business document suffices, as shown in the first highlighted code section. However, you might also need to provide totals by customer, which could be a document similar to a monthly statement, as well as grand totals, as shown in the second and third highlighted sections.

#### **VB 9.0**

```
Private Sub btnAggregates_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnAggregates.Click
    ' Create hierarchical Orders and Order Details
    ' by Customer -> Orders -> Order_Details associations
    ' with order summary and orders by customer summary
    Dim CustomerOrders = _
    <Customers>
        <%= From c In CustomerList _
            Select _
            <Customer>
                <CompanyName CustomerID=<%= c.CustomerID %>>
                    <%= c.CompanyName %></CompanyName>
                <City><%= c.City %></City>
                <Country><%= c.Country %></Country>
            <Orders>
                <%= From o In c.Orders _
                    Order By o.OrderID Descending _
                    Select _
                    <Order CustomerID=<%= o.CustomerID %>>
                        <OrderID><%= o.OrderID %></OrderID>
                        ' ... Omitted for brevity
                        <ShipToName><%= o.ShipName %></ShipToName>
                        <ShipToCountry><%= o.ShipCountry %></ShipToCountry>
                    <Details>
                        <%= From d In o.Order_Details _
                            Let s = CDec(d.Quantity) * CDec(d.UnitPrice) * _
                                (1 - CDec(d.Discount)) _
                            Select _
                            <Order_Detail OrderID=<%= d.OrderID %>>
                                <Quantity><%= d.Quantity %></Quantity>
                                ' ... Omitted for brevity
                                <Extended>
                                    <%= String.Format("{0:c}", s) %>
                                </Extended>
                            </Order_Detail> %
                        </Details>
```

## Chapter 7: Manipulating Documents with LINQ to XML

```
<OrderItems><%= o.Order_Details.Count %></OrderItems>
<OrderSubtotal> _
<%= String.Format("{0:c}", _
(From t In o.Order_Details _
Select CDec(t.Quantity) * CDec(t.UnitPrice) * _
(1 - CDec(t.Discount))).Sum()) %>
</OrderSubtotal>
<PrepaidFreight> _
<%= String.Format("{0:c}", o.Freight.Value) %>
</PrepaidFreight>
<OrderTotal> _
<%= String.Format("{0:c}", o.Freight + _
(From t In o.Order_Details _
Select CDec(t.Quantity) * CDec(t.UnitPrice) * _
(1 - CDec(t.Discount))).Sum()) %>
</OrderTotal>
</Order> %
</Orders>
<CustomerOrders><%= c.Orders.Count() %></CustomerOrders>
<OrdersSubtotal>
<%= String.Format("{0:c}", Aggregate o _
In c.Orders Into Sum(Aggregate d In o.Order_Details _
Into Sum(d.Quantity * d.UnitPrice * (1 - d.Discount))) %>
</OrdersSubtotal>
<OrdersFreight>
<%= String.Format("{0:c}", Aggregate f In c.Orders _
Into Sum(f.Freight)) %>
</OrdersFreight>
<OrdersTotal>
<%= String.Format("{0:c}", (From f In c.Orders _
Select f.Freight).Sum() + Aggregate o In c.Orders _
Into Sum(Aggregate d In o.Order_Details _
Into Sum(d.Quantity * d.UnitPrice * _
1 - d.Discount))) %>
</OrdersTotal>
</Customer> %
<GrandTotals>
<TotalOrders><%= OrderList.Count %></TotalOrders>
<TotalNetSales>
<%= String.Format("{0:c}", Aggregate d In Order_DetailList _
Into Sum(d.Quantity * d.UnitPrice * (1 - d.Discount))) %>
</TotalNetSales>
<AverageNetSale>
<%= String.Format("{0:c}", Aggregate o In OrderList _
Into Average(Aggregate d In o.Order_Details _
Into Sum(d.Quantity * d.UnitPrice * (1 - d.Discount))) %>
</AverageNetSale>
<TotalFreight>
<%= String.Format("{0:c}", Aggregate o In OrderList _
Into Sum(o.Freight)) %>
</TotalFreight>
<AverageFreight>
```

## Part III: Applying Domain-Specific LINQ Implementations

```
<%= String.Format("{0:c}", Aggregate o In OrderList _  
    Into Average(o.Freight)) %>  
</AverageFreight>  
<TotalGrossSales>  
    <%= String.Format("{0:c}", (Aggregate o In OrderList _  
        Into Sum(o.Freight)) + Aggregate o In OrderList _  
        Into Sum(Aggregate d In o.Order_Details _  
            Into Sum(d.Quantity * d.UnitPrice * (1 - d.Discount)))) %>  
</TotalGrossSales>  
</GrandTotals>  
</Customers>  
End Sub
```

*Notice the repetition of aggregation code in expressions for OrderSubtotal and OrderTotal, OrdersSubtotal and OrdersTotal, and TotalNetSales and TotalGrossSales elements. Let variables are scoped to their own expression hole, so they can't be used in successive calculations.*

Figure 7-6 shows IE 7 displaying part of the last order and all aggregates for the last member of the CustomerList data source, as well as grand totals for the 100 members of the OrderList object.



Figure 7-6

## Working with XML Namespaces and Local Names

XML Names consist of *namespace names*, optional *namespace prefixes*, and *local names* of elements and attributes, as defined in the W3C “Namespaces in XML 1.0 (Second Edition)” recommendation. The purpose of namespaces is to avoid conflicts when the same local name is contained in more than one vocabulary or has multiple meanings within an XML document. *Expanded names* consist of a namespace name in the form of a Universal Resource Identifier (URI) separated by a virgule (/) from a local name; an XName object represents an expanded name.

*The use of relative URIs as XML namespace names is deprecated.*

The most common URI for XML namespaces is a Web-style URL, as highlighted in the following example of a response from an ADO.NET Data Service in the Atom 1.0 syndication format to a request for all Northwind Orders:

### XML

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<feed xml:base="http://localhost:50539/Northwind.svc/">
  xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb"
  xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/metadata"
  xmlns="http://www.w3.org/2005/Atom">
  <id>http://localhost:50539/Northwind.svc/Orders</id>
  <updated />
  <title>Orders</title>
  <link rel="self" href="Orders/" title="Orders" />
  <entry adsm:type="NorthwindModel.Orders">
    <id>http://localhost:50539/Northwind.svc/Orders(10248)</id>
    <updated />
    <title />
    <author>
      <name />
    </author>
    <link rel="edit" href="Orders(10248)" title="Orders" />
    <content type="application/xml">
      <ads:OrderID adsm:type="Int32">10248</ads:OrderID>
      <ads:OrderDate adsm:type="Nullable`1[System.DateTime]">
        2005-07-04T00:00:00
      </ads:OrderDate>
      <ads:RequiredDate adsm:type="Nullable`1[System.DateTime]">
        2005-08-01T00:00:00
      </ads:RequiredDate>
      <ads:ShippedDate adsm:type="Nullable`1[System.DateTime]">
        2005-07-16T00:00:00
      </ads:ShippedDate>
      <ads:Freight adsm:type="Nullable`1[System.Decimal]">32.3800</ads:Freight>
      <ads:ShipName>Vins et alcools Chevalier</ads:ShipName>
      <ads:ShipAddress>59 rue de l'Abbaye</ads:ShipAddress>
      <ads:ShipCity>Reims</ads:ShipCity>
```

## Part III: Applying Domain-Specific LINQ Implementations

---

```
<ads:ShipRegion ads:null="true" />
<ads:ShipPostalCode>51100</ads:ShipPostalCode>
<ads:ShipCountry>France</ads:ShipCountry>
</content>
<link rel="related" title="Customers"
      href="Orders(10248)/Customers" type="application/atom+xml;type=entry" />
<link rel="related" title="Employees"
      href="Orders(10248)/Employees" type="application/atom+xml;type=entry" />
<link rel="related" title="Order_Details"
      href="Orders(10248)/Order_Details" type="application/atom+xml;type=feed" />
<link rel="related" title="Shippers"
      href="Orders(10248)/Shippers" type="application/atom+xml;type=entry" />
</entry>
</feed>
```

The preceding InfoSet, in which namespace prefixes are set bold, is missing elements for the Northwind Orders table's CustomerID, EmployeeID, and ShipVia fields. The Entity Framework is the default data source for ADO.NET Data Services; its Entity Data Model replaces foreign-key field values of many:1 associations with EntityReferences, which become `<link rel="related" ...>` elements of the single-instance entry type. The InfoSet represents 1:many associations (EntitySets) by similar elements of the multiple-instance feed types.

In the preceding example, `xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb"` and `xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/metadata"` are namespace declarations for the ads and adsm namespace prefixes; the URLs don't correspond to live Web locations. The string representation of the expanded name (XName) of the `<ads:OrderID>` element is `{http://schemas.microsoft.com/ado/2007/08/dataweb}OrderID`.

The `xmlns="http://www.w3.org/2005/Atom"` namespace specifies the *default namespace*, which applies to element and attribute names without a prefix; this URL opens the W3C/IETF's Atom Syndication Format Namespace page.

*XML documents in Atom 1.0 syndication format generated by ADO.NET Data Services for the Northwind sample database are used in several of this chapter's examples because they contain multiple namespaces, a variety of data types, and a moderately complex structure. Sample Atom 1.0 XML syndication files are located in the \WROX\ADONET\Chapter07\Data\AtomDS folder.*

LINQ to XML expands *all* XML namespace prefix references to local XNames during document manipulation; therefore, the full XName replaces *every* XML namespace prefix instance in the InfoSet. The expanded versions of this chapter's examples have at least twice the number of characters as the corresponding original versions. Saved LINQ to XML query results include the expanded namespaces unless your code adds the namespaces during document composition. In the latter case, the compiler caches the namespace names you add and doesn't expand them when you apply the `ToString()` method to an `XDocument` or `XElement` object.

C# 3.0 and VB 9.0 differ significantly in the way they handle LINQ to XML namespaces and namespace prefixes, as you'll see in the next few sections.

### XML Namespaces in C# 3.0

VS 2008's "Namespaces Overview (LINQ to XML)" online help topic claims that C# 3.0 "simplifies XML names by removing the requirement that developers use XML prefixes." Using XML prefixes is so ingrained in XLM developers that this claim obviously is a euphemism for "C# doesn't handle namespace prefixes as one would expect" or "functional construction treats namespaces strangely." For starters, the VB Editor integrates IntelliSense with namespace prefixes, as shown in the next section; C# 3.0's Editor doesn't provide IntelliSense.

*The XmlNamespacesCS.sln project in the \WROX\ADONET\Chapter07\CS folder contains C# code examples for LINQ to XML queries and functional construction of XDocuments and XElements with multiple namespaces.*

### Namespaces in C# LINQ to XML Queries

You must declare your namespaces and their prefixes prior to referencing them in queries or functional construction statements. Instead of using the traditional colon separator ("nsName:elementName"), functional construction uses a plus sign (nsName + "elementName"), as shown emphasized in the following code that lists U.S. orders in reverse chronological order from the Orders.xml InfoSet shown in the preceding section:

#### C# 3.0

```
private void btnUSOrders_Click(object sender, EventArgs e)
{
    // Return an XML InfoSet in Atom format of US orders in descending date order
    string fileOrders = @"\\WROX\\ADONET\\Chapter07\\Data\\AtomDS\\Orders.xml";
    XDocument xdOrders = XDocument.Load(fileOrders,
        LoadOptions.PreserveWhitespace);

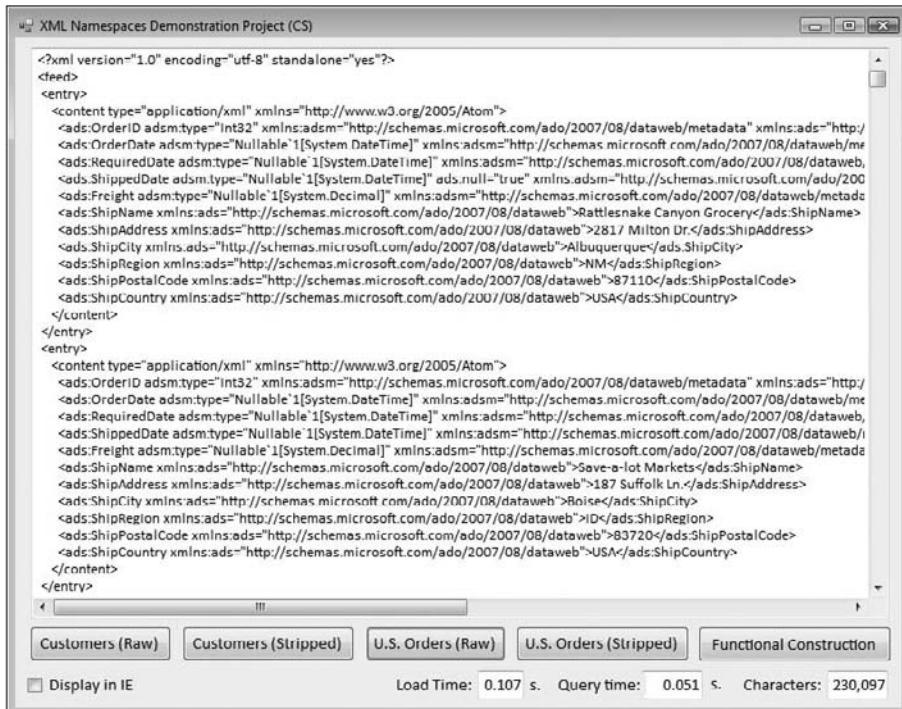
    XNamespace atom = "http://www.w3.org/2005/Atom";
    XNamespace ads = "http://schemas.microsoft.com/ado/2007/08/dataweb";
    XNamespace adsm = "http://schemas.microsoft.com/ado/2007/08/dataweb/metadata";

    var orders = from o in xdOrders.Descendants(atom + "content")
                 where o.Element(ads + "ShipCountry").Value == "USA"
                 orderby o.Element(ads + "OrderDate").Value descending
                 select o;

    // Create the StringBuilder for the InfoSet
    sbXml.Length = 0;
    sbXml.Append("<?xml version=\"1.0\" encoding=\"utf-8\""
                 + "standalone=\"yes\"?>\r\n");
    sbXml.Append("<feed>\r\n");
    sbXml.Append("<feed>\r\n");
    foreach (var c in custs)
    {
        sbXml.Append("  <entry>\r\n");
        sbXml.Append("    " + c.ToString() + "\r\n");
        sbXml.Append("  </entry>\r\n");
    }
    sbXml.Append("</feed>\r\n");      sbXml.Append("</feed>\r\n");
}
```

## Part III: Applying Domain-Specific LINQ Implementations

Figure 7-7 shows the XmlNamespacesCS.sln project displaying the result of the preceding query with the global ads and adsm namespaces defined in the `<feed>` element expanded as local namespaces for members of the `<content>` group. The default atom namespace is expanded as a `<content>` attribute. Obviously, adding local expanded namespaces decreases the readability of the document greatly.



The screenshot shows a Windows application window titled "XML Namespaces Demonstration Project (CS)". Inside, there is a large text area displaying XML code. Below the text area are several buttons: "Customers (Raw)", "Customers (Stripped)", "U.S. Orders (Raw)", "U.S. Orders (Stripped)", and "Functional Construction". At the bottom, there are checkboxes for "Display in IE" and "Load Time: 0.107 s. Query time: 0.051 s. Characters: 230,097".

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<feed>
<entry>
<content type="application/xml" xmlns="http://www.w3.org/2005/Atom">
<ads:OrderID adsm:type="int32" xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/metadata" xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb/me</ads:OrderID>
<ads:OrderDate adsm:type="Nullable`1[System.DateTime]" xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/me</ads:OrderDate>
<ads:RequiredDate adsm:type="Nullable`1[System.DateTime]" xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/me</ads:RequiredDate>
<ads:ShippedDate adsm:type="Nullable`1[System.DateTime]" xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/me</ads:ShippedDate>
<ads:Freight adsm:type="Nullable`1[System.Decimal]" xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/mctads</ads:Freight>
<ads:ShipName xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb">Rattlesnake Canyon Grocery</ads:ShipName>
<ads:ShipAddress xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb">2817 Milton Dr.</ads:ShipAddress>
<ads:ShipCity xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb">Albuquerque</ads:ShipCity>
<ads:ShipRegion xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb">NM</ads:ShipRegion>
<ads:ShipPostalCode xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb">87110</ads:ShipPostalCode>
<ads:ShipCountry xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb">USA</ads:ShipCountry>
</content>
</entry>
<entry>
<content type="application/xml" xmlns="http://www.w3.org/2005/Atom">
<ads:OrderID adsm:type="int32" xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/metadata" xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb/me</ads:OrderID>
<ads:OrderDate adsm:type="Nullable`1[System.DateTime]" xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/me</ads:OrderDate>
<ads:RequiredDate adsm:type="Nullable`1[System.DateTime]" xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/me</ads:RequiredDate>
<ads:ShippedDate adsm:type="Nullable`1[System.DateTime]" xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/me</ads:ShippedDate>
<ads:Freight adsm:type="Nullable`1[System.Decimal]" xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/metada</ads:Freight>
<ads:ShipName xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb">Save-a-lot Markets</ads:ShipName>
<ads:ShipAddress xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb">187 Suffolk Ln.</ads:ShipAddress>
<ads:ShipCity xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb">Noise</ads:ShipCity>
<ads:ShipRegion xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb">D</ads:ShipRegion>
<ads:ShipPostalCode xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb">83720</ads:ShipPostalCode>
<ads:ShipCountry xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb">USA</ads:ShipCountry>
</content>
</entry>

```

Figure 7-7

## Removing Expanded Namespaces from C# Queries

Although expanded prefixed namespaces appear as conventional attributes of their associated `XElement`, you can't remove them with code, as in the following, in the `foreach` loop:

### C# 3.0

```
foreach ( XElement x in o.Elements() )
{
    int Ctr = x.Attributes().Count();
    foreach ( XAttribute a in x.Attributes() )
    {
        if ( a.IsNamespaceDeclaration )
            a.Remove();
    }
}
```

`Ctr = 1` for value types because it doesn't recognize the `adsm:type="int32"` or `adsm:type="Nullable`1[System.Decimal]"` as an attribute. Strings and local namespace declarations don't increment `Ctr`.

## Chapter 7: Manipulating Documents with LINQ to XML

Instead, you must operate on the string obtained by applying the `ToString()` and `Replace()` operators to the `XElements`, as in this example:

### C# 3.0

```
// Namespace attribute strings required for removal
string strDefaultNs = @”xmlns="" +
    orders.ElementAt(0).GetDefaultNamespace().NamespaceName + @""";
string strAdsNs = @”xmlns:ads="" +
    orders.ElementAt(0).GetNamespaceOfPrefix("ads").NamespaceName + @""";
string strAdsmNs = @”xmlns:adsm="" +
    orders.ElementAt(0).GetNamespaceOfPrefix("adsm").NamespaceName + @""";
// ...
// You must remove the namespaces from the string representation
foreach ( XElement x in o.Elements())
{
    string strOrder = "      " + o.ToString();
    strOrder = strOrder.Replace(strDefaultNs, " ")
        .Replace(strAdsmNs + " ", " ")
        .Replace(strAdsNs, " ");
    sbXml.Append(" <entry>\r\n");
    sbXml.Append("      " + o.ToString() + "\r\n");
    sbXml.Append(" </entry>\r\n");
}
```

The preceding code, which returns the original version of the `<content>` subelements shown in Figure 7-8, is part of the `XmlNamespacesCS.sln` project's `btnOrdersStripped_Click` event handler.

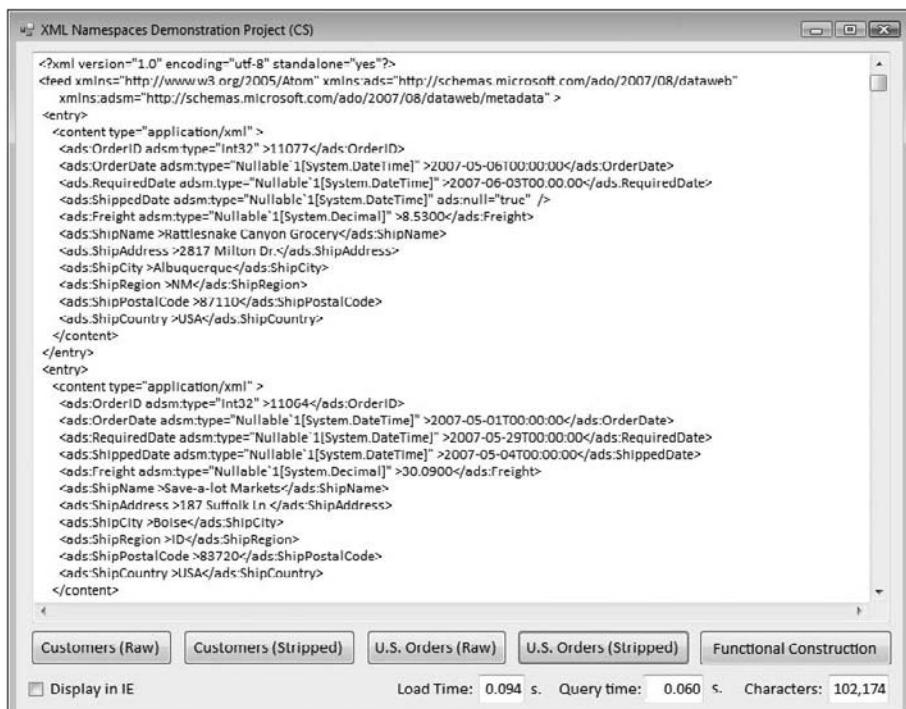


Figure 7-8

## Part III: Applying Domain-Specific LINQ Implementations

---

### Functionally Constructing C# XDocuments with Multiple Namespaces

An alternative to removing namespaces with the previous section's method is to regenerate the document with functional construction. You must define and add the default namespace to the root element (with atom + "feed" for this example) and then overwrite it with a new XAttribute("xmlns", "namespaceName") statement. Additional prefixed global or group namespace declarations require new XAttribute(XNamespace.Xmlns + "prefix", "namespaceName") expressions.

*An XAttribute with "xmlns" as its name is a special attribute that's not in any namespace. The document then serializes with the special attribute's namespace as the default namespace.*

Namespace declarations in the following code for the Functional Construction button's Click event handler are highlighted:

#### C# 3.0

```
// Generate new document with namespaces using functional construction
private void btnFuncConst_Click(object sender, EventArgs e)
{
    string fileOrders = @"\WROX\ADONET\Chapter07\Data\AtomDS\Orders.xml";
    XDocument xdOrders = XDocument.Load(fileOrders,
                                         LoadOptions.PreserveWhitespace);

    // Declare the named default namespace
    XNamespace atom = "http://www.w3.org/2005/Atom";

    var ordersDoc = new XDocument(
        new XDeclaration("1.0", "utf-8", "yes"),
        // Add default namespace
        new XElement(atom + "feed",
            // Overwrite as default namespace
            new XAttribute("xmlns", "http://www.w3.org/2005/Atom"),
            // Add other global namespaces
            new XAttribute(XNamespace.Xmlns + "ads",
                "http://schemas.microsoft.com/ado/2007/08/dataweb"),
            new XAttribute(XNamespace.Xmlns + "adsm",
                "http://schemas.microsoft.com/ado/2007/08/dataweb/metadata"),
            from o in xdOrders.Descendants(atom + "content")
            where o.Element(ads + "ShipCountry").Value == "USA"
            orderby o.Element(ads + "OrderDate").Value descending
            select new XElement("entry", new XElement("content",
                new XAttribute("type", "application/xml"),
                new XElement(ads + "OrderID",
                    o.Element(ads + "OrderID").Value,
                    new XAttribute(adsm + "type", "int32"))),
                new XElement(ads + "OrderDate",
                    o.Element(ads + "OrderDate").Value,
                    new XAttribute(adsm + "type", "Nullable`1[System.DateTime]")),
                new XElement(ads + "RequiredDate",
                    o.Element(ads + "RequiredDate").Value,
                    new XAttribute(adsm + "type", "Nullable`1[System.DateTime]"))),
            new XElement(ads + "TotalOrderCost",
                o.Element(ads + "TotalOrderCost").Value,
                new XAttribute(adsm + "type", "decimal"))
        )
    );
}
```

## Chapter 7: Manipulating Documents with LINQ to XML

```
        new XElement(ads + "ShippedDate",
            o.Element(ads + "ShippedDate").Value,
            new XAttribute(adsm + "type", "Nullable`1[System.DateTime]")),
        new XElement(ads + "Freight", o.Element(ads + "Freight").Value,
            new XAttribute(adsm + "type", "Nullable`1[System.Decimal]")),
        new XElement(ads + "ShipName",
            o.Element(ads + "ShipName").Value),
        new XElement(ads + "ShipAddress",
            o.Element(ads + "ShipAddress").Value),
        new XElement(ads + "ShipCity",
            o.Element(ads + "ShipCity").Value),
        new XElement(ads + "ShipRegion",
            o.Element(ads + "ShipRegion").Value),
        new XElement(ads + "ShipPostalCode",
            o.Element(ads + "ShipPostalCode").Value),
        new XElement(ads + "ShipCountry",
            o.Element(ads + "ShipCountry").Value)
    ))))
;
}
```

The preceding transformation code duplicates the original `XDocument`'s content elements except for the `item` element's mysterious `xmlns=""` empty namespace attribute, as illustrated by Figure 7-9.

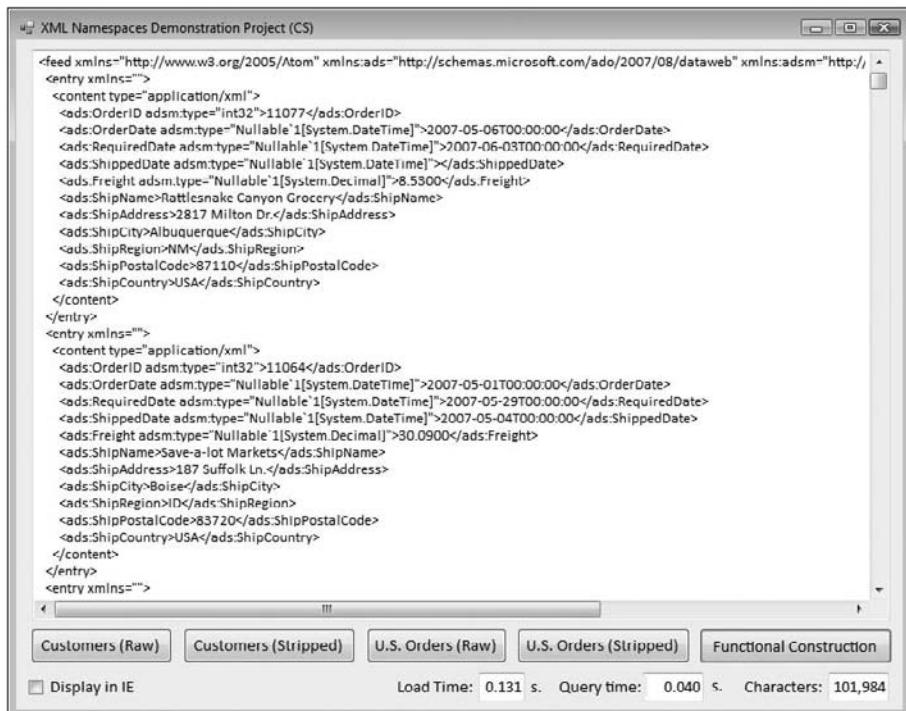


Figure 7-9

## Part III: Applying Domain-Specific LINQ Implementations

---

Minor modifications to the preceding example's code enables generating namespace-rich Infosets from other data sources, such as LINQ to SQL, LINQ to Entities, comma-separated or tab-separated text files, or `SqlDataReaders`.

### **XML Namespaces in VB 9.0**

VB enables adding global namespaces as XML literal elements and, as noted earlier, supports functional construction of `XDocument` and `XElement` objects. Supporting IntelliSense with multiple namespaces requires more schema nuances than the VS 2008 XML Editor's default schema inference feature provides, so you must download and use the XML To Schema template described in the next section.

*The XmlNamespacesVB.sln project in the \WROX\ADONET\Chapter07\CS folder contains VB code examples for LINQ to XML queries, as well as XML literal composition and functional construction of XDocuments having multiple namespaces.*

#### **Enabling IntelliSense for Namespaces**

As mentioned in the earlier "Inferring a Schema and Enabling IntelliSense for VB Queries" section, enabling IntelliSense for LINQ to XML queries against Infosets having namespaces requires the Schema Tool for Visual Basic 2008 add-in (also known as the XML to Schema Wizard). This tool, which is included in VS 2008 SP1, improves handling of multiple schemas for different namespaces, consolidates schemas for multiple similar documents, and refines schema generation to prevent naming collisions in complex documents.

To infer and associate the schema with the XML files that you've added to your project, do the following:

- 1.** Open the Add New Item dialog, select the XML To Schema icon to open the Infer XML Schema from XML Documents dialog, and type a name for your schema file(s), such as `AtomXmlToSchema` for this example.
- 2.** Assuming that XML files are your data source, click the Add From Files button to open the Add XML Files Dialog, navigate to the appropriate folder (\WROX\ADONET\Chapter07\Data\AtomDS for this example) select the files to add (all for this example), and click the Open button to close the dialog. The Infer XML Schema from XML Documents dialog now appears, as shown in Figure 7-7.
- 3.** Click OK to add the inferred schemas to the project folder, and close the Schema Tool dialog.

All Atom XML files created by ADO.NET Data Services infer a common set of four schemas. The `AtomXmlToSchema1.xsd` primary schema for the Atom 1.0 namespace, <http://www.w3.org/2005/Atom>, imports three related schemas: `AtomXmlToSchema.xsd` for <http://www.w3.org/XML/1998/namespace>, `AtomXmlToSchema2.xsd` for <http://schemas.microsoft.com/ado/2007/08/dataweb/>

## Chapter 7: Manipulating Documents with LINQ to XML

metadata, and AtomXmlToSchema3.xsd for `http://schemas.microsoft.com/ado/2007/08/dataweb`. AtomXmlToSchema1's section for the `<content>` group defines a sequence that contains the names of all 44 elements of the `<content>` groups from the six Atom-format XML documents shown in Figure 7-10.

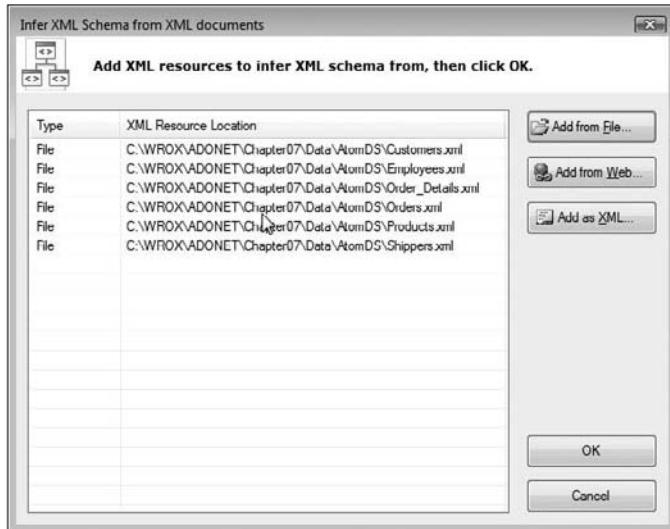


Figure 7-10

The `XmlNamespacesVB.sln` in the `\WROX\ADONET\Chapter07\VB` folder contains VB code examples for LINQ to XML queries and XML literal composition of `XDocuments` and `XElements` with namespaces.

Figure 7-11 shows the VB editor's IntelliSense for descendants (top), elements (middle), and prefixed element names (bottom).

## Part III: Applying Domain-Specific LINQ Implementations

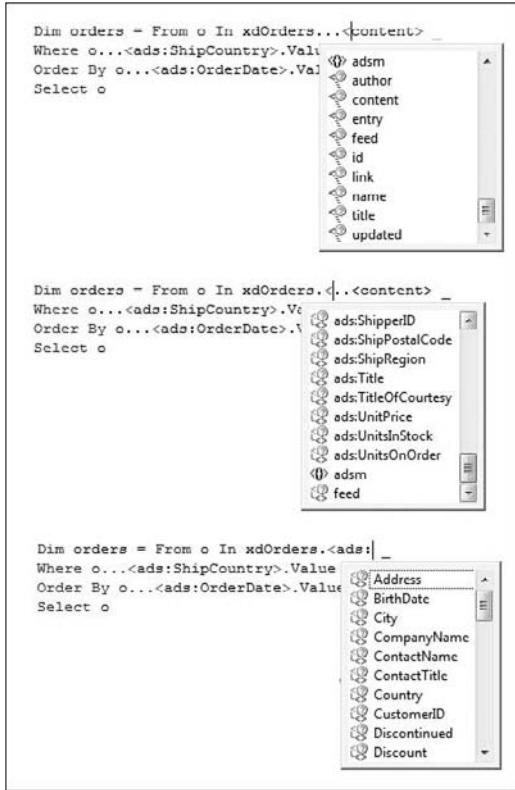


Figure 7-11

The question mark symbols in the IntelliSense lists, which indicate that the editor hasn't yet determined the data type for the item, turn to check marks after you build the associated code successfully.

### Multiple Namespaces in XML Literal Queries

The VB compiler requires namespaces for literal XML query composition to be imported with directives such as the following:

#### VB 9.0

```
Imports <xmllns="http://www.w3.org/2005/Atom">
Imports <xmllns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb">
Imports <xmllns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/metadata">
```

The Imports directives replace the equivalent XNamespace variable declarations shown in the examples of the earlier “Namespaces in C# LINQ to XML Queries” section. Otherwise, porting C# query code to VB, and vice versa, is straightforward. The only significant namespace-related changes are adding namespace prefixes, as show in bold below:

### VB 9.0

```
Private Sub btnUSOrders_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnUSOrders.Click
    ' Return an XML InfoSet containing US orders in descending date order
    txtXML.Text = ""

    Dim fileOrders As String = "\WROX\ADONET\Chapter07\Data\AtomDS\Orders.xml"
    Dim xdOrders As XDocument = XDocument.Load(fileOrders, _
        LoadOptions.PreserveWhitespace)

    Dim orders = From o In xdOrders...<content> _
        Where o...<ads:ShipCountry>.Value = "USA" _
        Order By o...<ads:OrderDate>.Value Descending _
        Select o

    ' Create the StringBuilder for the InfoSet
    sbXml.Length = 0
    sbXml.Append("<?xml version=""1.0"" encoding=""utf-8"" standalone=""yes""?>" _ & vbCrLf)
    sbXml.Append("<feed>" & vbCrLf)
    For Each o In orders
        sbXml.Append("  <entry>" & vbCrLf)
        sbXml.Append("    " & o.ToString() & vbCrLf)
        sbXml.Append("  </entry>" & vbCrLf)
    Next o
    sbXml.Append("</feed>" & vbCrLf)

    txtXML.Text = sbXml.ToString()
    txtChars.Text = sbXml.Length.ToString("#,##0")
    DisplayInIE("Orders.xml")
End Sub
```

There's no difference between the output of the `btnOrders_Click` and `btnOrdersStripped_Click` event handlers between C# and VB versions. The code to strip the expanded namespaces from the `XDocument` output involves minor syntactic changes only.

## Part III: Applying Domain-Specific LINQ Implementations

---

### **Transforming Documents That Have Multiple Namespaces, with Literal XML Code**

The following code transforms the Atom-formatted source document to an output document that contains only <content> groups with namespace prefixes set bold:

#### **VB 9.0**

```
Private Sub btnLiteral_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnLiteral.Click
    ' Return an XML InfoSet containing US orders in descending date order
    txtXML.Text = ""

    Dim fileOrders As String = "\WROX\ADONET\Chapter07\Data\AtomDS\Orders.xml"
    Dim xdOrders As XDocument = XDocument.Load(fileOrders, _
                                                LoadOptions.PreserveWhitespace)

    Dim Orders As XDocument = Nothing
    Orders = _
        <?xml version="1.0" encoding="utf-8" standalone="yes"?>
        <feed xmlns="http://www.w3.org/2005/Atom"
              xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb"
              xmlns:adsm="http://schemas.microsoft.com/" & _
              "ado/2007/08/dataweb/metadata">
            <%= From o In xdOrders...<content> _
                Where o...<ads:ShipCountry>.Value = "USA" _ 
                Order By o...<ads:OrderDate>.Value Descending _ 
                Select New XElement( _
                    <entry>
                        <content type="application/xml">
                            <ads:OrderID adsm:type="Int32">
                                <%= o...<ads:OrderID>.Value %>
                            </ads:OrderID>
                            <ads:OrderDate adsm:type="Nullable`1[System.DateTime]">
                                <%= o...<ads:OrderDate>.Value %>
                            </ads:OrderDate>
                            <ads:RequiredDate adsm:type="Nullable`1[System.DateTime]">
                                <%= o...<ads:RequiredDate>.Value %>
                            </ads:RequiredDate>
                            <ads:ShippedDate adsm:type="Nullable`1[System.DateTime]">
                                <%= o...<ads:ShippedDate>.Value %>
                            </ads:ShippedDate>
                            <ads:Freight adsm:type="Nullable`1[System.Decimal]">
                                <%= o...<ads:Freight>.Value %>
                            </ads:Freight>
                            <ads:ShipName>
                                <%= o...<ads:ShipName>.Value %>
                            </ads:ShipName>
                            <ads:ShipAddress>
                                <%= o...<ads:ShipAddress>.Value %>
                            </ads:ShipAddress>
                            <ads:ShipCity>
                                <%= o...<ads:ShipCity>.Value %>
                            </ads:ShipCity>
                            <ads:ShipRegion>
```

## Chapter 7: Manipulating Documents with LINQ to XML

```
<%= o...<ads:ShipRegion>.Value %>
</ads:ShipRegion>
<ads:ShipPostalCode>
    <%= o...<ads:ShipPostalCode>.Value %>
</ads:ShipPostalCode>
<ads:ShipCountry>
    <%= o...<ads:ShipCountry>.Value %>
</ads:ShipCountry>
</content>
</entry>) _
%>>
</feed>
DisplayInIE("Orders.xml")
End Sub
```

The output of literal XML composition with namespaces unexpectedly duplicates global default namespaces as attributes of an ancestor of content-containing elements, `<entry>` for this example, as shown in Figure 7-12.

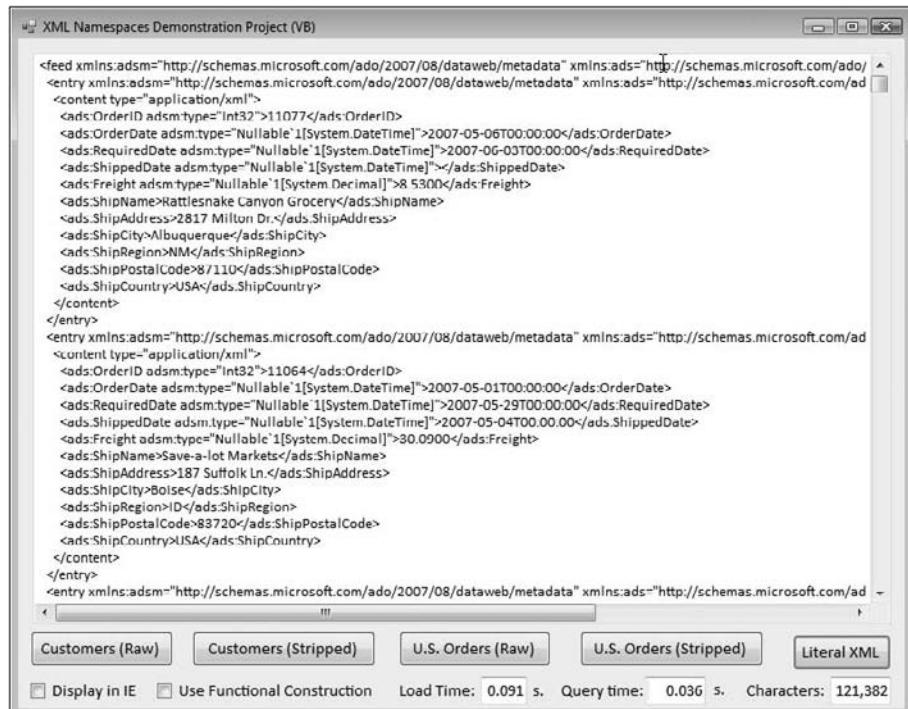


Figure 7-12

## Part III: Applying Domain-Specific LINQ Implementations

---

The element with all four attributes is <content type="application/xml" xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/metadata" xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb" xmlns="http://www.w3.org/2005/Atom">; the three namespace attributes are redundant.

## Performing Heterogeneous Joins and Lookup Operations

A LINQ feature that interested early adopters and technical analysts alike was the capability to create or emulate joins between heterogeneous object types — typically XML documents, objects, and relational tables — and take advantage of a single query language for all objects. The need to use LINQ to XML extension methods to navigate Infosets and to use VB 9.0's literal XML data type or XDocument() / XElement() methods to compose Infosets makes the "single query language" claim a bit of a stretch. LINQ to Objects, LINQ to SQL, LINQ to Entities, and most other third-party LINQ implementations use the same or slightly modified LINQ query expressions.

*The examples for the following section sections are HeterogeneousXmlJoinsCS.sln and HeterogeneousXmlJoinsVB.sln in the \WROX\ADONET\Chapter07\CS and . . . \VB folders. The VB version demonstrates the significant differences in namespace processing when composing Infosets with literal XML and functional construction code.*

When you compose Infosets from multiple documents, object graphs or relational tables, *joins* are best suited to *flat* operations, such as populating elements representing many:1 relationships, while *lookup operations* handle *hierarchical additions* for 1:many relationships. However, there is little or no difference in query execution with Join or Where clauses in most cases.

## Using Lookup Operations to Add Child Element Groups

A simple demonstration of a lookup operation to add multiple lower members of an XML document hierarchy is adding Order\_Details elements to the content element of the Atom 1.0-formatted Infoset used in the preceding sections. For this example, the output document's abbreviated structure is:

```
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb"
      xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/metadata">
  <entry>
    <content type="application/xml">
      <Order>
        <ads:OrderID adsm:type="int32">11006</ads:OrderID>
        <!-- ... -->
        <ads:ShipCountry>USA</ads:ShipCountry>
        <Order_Details>
          <Order_Detail>
            <ads:OrderID adsm:type="System.Int32">11006</ads:OrderID>
            <ads:ProductID adsm:type="System.Int32">1</ads:ProductID>
            <ads:Quantity adsm:type="System.Int16">8</ads:Quantity>
            <ads:UnitPrice adsm:type="System.Decimal">18.0000</ads:UnitPrice>
```

```
<ads:Discount adsm:type="System.Single">0</ads:Discount>
</Order_Detail>
<Order_Detail>
    <!-- ... -->
</Order_Detail>
    <!-- ... -->
</Order_Details>
</Order>
</content>
</entry>
```

Following are examples of abbreviated C# and VB code (emphasized) to add the individual Order\_Details elements from the Atom 1.0-formatted Order\_Details.xml document:

## C# 3.0 Functional Construction

```
private void btnOrder_DetailsLookup_Click(object sender, EventArgs e)
{
    XDocument xdOrders = XDocument.Load(strDataPath + "Orders.xml",
                                         LoadOptions.PreserveWhitespace);
    XDocument xdDetails = XDocument.Load(strDataPath + "Order_Details.xml",
                                         LoadOptions.PreserveWhitespace);

    XNamespace atom = "http://www.w3.org/2005/Atom";
    XNamespace ads = "http://schemas.microsoft.com/ado/2007/08/dataweb";
    XNamespace adsm = "http://schemas.microsoft.com/ado/2007/08/dataweb/metadata";

    XDocument Orders = new XDocument(new XDeclaration("1.0", "utf-8", "yes"),
                                      new XElement(atom + "feed",
                                                  new XAttribute("xmlns", "http://www.w3.org/2005/Atom"),
                                                  new XAttribute(XNamespace.Xmlns + "ads",
                                                               "http://schemas.microsoft.com/ado/2007/08/dataweb"),
                                                  new XAttribute(XNamespace.Xmlns + "adsm",
                                                               "http://schemas.microsoft.com/ado/2007/08/dataweb/metadata"),
                                                  (from o in xdOrders.Descendants(atom + "content")
                                                   where o.Element(ads + "ShipCountry").Value == "USA"
                                                   orderby o.Element(ads + "OrderDate").Value descending
                                                   select new XElement("entry",
                                                       new XElement("content", new XAttribute("type",
                                                               "application/xml"),
                                                       new XElement("Order", new XElement(ads + "OrderID",
                                                               o.Element(ads + "OrderID").Value,
                                                               new XAttribute(adsm + "type", "int32")),
                                                       <!-- ... -->
                                                       new XElement(ads + "ShipCountry",
                                                               o.Element(ads + "ShipCountry").Value),
                                                       new XElement("Order_Details",
                                                       from d in xdDetails.Descendants(atom + "content")
                                                       where d.Element(ads + "OrderID").Value ==
                                                       o.Element(ads + "OrderID").Value
                                                       select new XElement("Order_Detail",
```

## Part III: Applying Domain-Specific LINQ Implementations

```
        new XElement(ads + "OrderID",
            d.Element(ads + "OrderID").Value,
            new XAttribute(adsm + "type",
                "System.Int32")),
        new XElement(ads + "ProductID",
            d.Element(ads + "ProductID").Value,
            new XAttribute(adsm + "type",
                "System.Int32")),
        new XElement(ads + "Quantity",
            d.Element(ads + "Quantity").Value,
            new XAttribute(adsm + "type",
                "System.Int16")),
        new XElement(ads + "UnitPrice",
            d.Element(ads + "UnitPrice").Value,
            new XAttribute(adsm + "type",
                "System.Decimal")),
        new XElement(ads + "Discount",
            d.Element(ads + "Discount").Value,
            new XAttribute(adsm + "type",
                "System.Single"))))))));
    }
```

The LINQ query for lookup or join operations is set bold in the preceding and following examples.

### VB 9.0 Literal XML

```
Private Sub btnOrder_DetailsLookup_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles btnProductsJoinXML.Click
    Dim xdOrders As XDocument = XDocument.Load(strDataPath & "Orders.xml", _
        LoadOptions.PreserveWhitespace)
    Dim xdDetails As XDocument = XDocument.Load(strDataPath & _
        "Order_Details.xml", LoadOptions.PreserveWhitespace)

    Dim Orders As XDocument = _
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:ads="http://schemas.microsoft.com/ado/2007/08/dataweb"
      xmlns:adsm="http://schemas.microsoft.com/ado/2007/08/dataweb/metadata"
      <%= From o In xdOrders...<content> _
          Where o...<ads:ShipCountry>.Value = "USA" _
          Order By o...<ads:OrderDate>.Value Descending _
          Select New XElement( _
<entry>
    <content type="application/xml">
        <Order>
            <ads:OrderID adsm:type="Int32">
                <%= o...<ads:OrderID>.Value %>
            </ads:OrderID>
            <!-- ... -->
            <ads:ShipCountry>
                <%= o...<ads:ShipCountry>.Value %>
            </ads:ShipCountry>
```

```
<Order_Details  
  <%= From d In xdDetails...<content> _  
    Where d...<ads:OrderID>.Value = o...<ads:OrderID>.Value _  
    Select New XElement( _  
      <Order_Detail>  
        <ads:OrderID adsm:type="Int32">  
          <%= d...<ads:OrderID>.Value %>  
        </ads:OrderID>  
        <ads:ProductID adsm:type="Int32">  
          <%= d...<ads:ProductID>.Value %>  
        </ads:ProductID>  
        <ads:Quantity adsm:type="Short">  
          <%= d...<ads:Quantity>.Value %>  
        </ads:Quantity>  
        <ads:QuantityPerUnit>  
          <%= p...<ads:QuantityPerUnit>.Value %>  
        </ads:QuantityPerUnit>  
        <ads:UnitPrice adsm:type="Decimal">  
          <%= d...<ads:UnitPrice>.Value %>  
        </ads:UnitPrice>  
        <ads:Discount adsm:type="Single">  
          <%= d...<ads:Discount>.Value %>  
        </ads:Discount>  
      </Order_Detail>) _  
    %>>  
  </Order_Details>  
</Order>  
</content>  
</entry>)  
%>>  
</feed>  
End Sub
```

Using the VB literal XML data type adds the spurious (duplicate) namespace declaration attributes shown in Figure 7-12 to each Order\_Detail element, as well as to the entry element.

### VB 9.0 Functional Construction

```
Private Sub btnOrder_DetailsLookup_Click(ByVal sender As System.Object,  
  ByVal e As System.EventArgs) Handles btnProductsJoinXML.Click  
  Dim xdOrders As XDocument = XDocument.Load(strDataPath & "Orders.xml", _  
    LoadOptions.PreserveWhitespace)  
  Dim xdDetails As XDocument = XDocument.Load(strDataPath & _  
    "Order_Details.xml", LoadOptions.PreserveWhitespace)  
  
  ' Local namespace variables required for functional construction  
  Dim atom As XNamespace = "http://www.w3.org/2005/Atom"  
  Dim ads As XNamespace = "http://schemas.microsoft.com/ado/2007/08/dataweb"  
  Dim adsm As XNamespace = _  
    "http://schemas.microsoft.com/ado/2007/08/dataweb/metadata"  
  
  Dim Orders As XDocument = New XDocument( _  
    New XDeclaration("1.0", "utf-8", "yes"), _
```

## Part III: Applying Domain-Specific LINQ Implementations

```
New XElement(atom + "feed", _  
    New XAttribute("xmlns", "http://www.w3.org/2005/Atom"), _  
    New XAttribute(XNamespace.Xmlns + "ads", _  
        "http://schemas.microsoft.com/ado/2007/08/dataweb"), _  
    New XAttribute(XNamespace.Xmlns + "adsm",  
        "http://schemas.microsoft.com/ado/2007/08/dataweb/metadata"), _  
    From o In xdOrders.Descendants(atom + "content") _  
    Where o.Element(ads + "ShipCountry").Value = "USA" _  
    Order By o.Element(ads + "OrderDate").Value Descending _  
    Select New XElement("entry", _  
        New XElement("content", _  
            New XAttribute("type", "application/xml"), New XElement("Order", _  
                New XElement(ads + "OrderID", _  
                    o.Element(ads + "OrderID").Value, _  
                    New XAttribute(adsm + "type", "int32")), _  
                <!-- ... -->  
                New XElement(ads + "ShipCountry", _  
                    o.Element(ads + "ShipCountry").Value), _  
                New XElement("Order_Details", _  
                    From d In xdDetails...<content> _  
                    Where d...<ads:OrderID>.Value = o...<ads:OrderID>.Value _  
                    Select New XElement("Order_Detail", _  
                        New XElement(ads + "OrderID", _  
                            d.Element(ads + "OrderID").Value, _  
                            New XAttribute(adsm + "type", "System.Int32")), _  
                        New XElement(ads + "ProductID", _  
                            d.Element(ads + "ProductID").Value, _  
                            New XAttribute(adsm + "type", "System.Int32")), _  
                        New XElement(ads + "Quantity", _  
                            d.Element(ads + "Quantity").Value, _  
                            New XAttribute(adsm + "type", "System.Int16")), _  
                        New XElement(ads + "QuantityPerUnit", _  
                            p.Element(ads + "QuantityPerUnit").Value), _  
                        New XElement(ads + "UnitPrice", _  
                            d.Element(ads + "UnitPrice").Value, _  
                            New XAttribute(adsm + "type", "System.Decimal")), _  
                        New XElement(ads + "Discount", _  
                            d.Element(ads + "Discount").Value, _  
                            New XAttribute(adsm + "type", "System.Single")) _  
                    ))))))))  
End Sub
```

Using the VB or C# functional construction approach adds a bogus xmlns="" namespace declaration attribute to the entry element. The VB functional construction example is present only to demonstrate that VB adds the extra empty namespace attribute also.

## Joining Documents to Insert Elements

Adding a new element based on an equi-join with a related document, Order\_Details.xml, for this example, requires adding the join `x` in `elements` on `outerKey.Value` equals

## Chapter 7: Manipulating Documents with LINQ to XML

*innerKey.Value* expression, as shown here for the added C# 3.0 statements (emphasized) of the *btnProductsJoinXML* event handler:

### C# 3.0 Functional Construction

```
<!-- ... -->
XDocument xdProducts = XDocument.Load(strDataPath + "Products.xml",
                                         LoadOptions.PreserveWhitespace);

<!-- ... -->
new XElement("Order_Details",
    from d in xdDetails.Descendants(atom + "content")
    where d.Element(ads + "OrderID").Value == o.Element(ads + "OrderID").Value
    join p in xdProducts.Descendants(atom + "content")
        on d.Element(ads + "ProductID").Value equals
            p.Element(ads + "ProductID").Value
    select new XElement("Order_Detail",
        new XElement(ads + "ProductID",
            d.Element(ads + "ProductID").Value,
            new XAttribute(adsm + "type", "System.Int32")),
        new XElement(ads + "ProductName",
            p.Element(ads + "ProductName").Value),
        <!-- ... -->
        new XElement(ads + "Discount",
            d.Element(ads + "Discount").Value,
            new XAttribute(adsm + "type", "System.Single"))))));
```

*Highlighted added code together with initial and terminal statements only appear in the preceding and following snippets for brevity.*

### VB 9.0 Literal XML

```
Dim xdProducts As XDocument = XDocument.Load(strDataPath & "Products.xml", _
                                              LoadOptions.PreserveWhitespace)

<!-- ... -->
<Order_Details
<%= From d In xdDetails...<content> _
    Where d...<ads:OrderID>.Value = o...<ads:OrderID>.Value _
    Join p In xdProducts...<content>
        On o...<ads:ProductID>.Value Equals d...<ads:ProductID>.Value _
    Select New XElement( _
        <Order_Detail>
            <ads:ProductID adsm:type="Int32">
                <%= d...<ads:ProductID>.Value %>
            </ads:ProductID>
            <ads:ProductName>
                <%= p...<ads:ProductName>.Value %>
            </ads:ProductName>
            <!-- ... -->
        </Order_Detail>) _
%>>
</Order_Details>
```

## Part III: Applying Domain-Specific LINQ Implementations

### Joining Documents and LINQ to SQL or LINQ to Object Entities

Code to insert elements from many:1-related LINQ to SQL or LINQ to Object entities is almost identical. The following example adds CategoryID, CategoryName, SupplierID, and SupplierName elements from a NorthwindDataContext or CategoryList<Category> and SupplierList<Supplier> collections in the btnSupplCatJoinSQL\_Click event handler:

#### C# 3.0 Functional Construction

```
<!-- ... -->
NorthwindDataContext dcNwind = new NorthwindDataContext();
<!-- ... -->
new XElement("Order_Details",
    from d in xdDetails.Descendants(atom + "content")
    where d.Element(ads + "OrderID").Value == o.Element(ads + "OrderID").Value
    join p in dcNwind.Products
        on d.Element(ads + "ProductID").Value equals p.ProductID.ToString()
    join c in dcNwind.Categories on p.CategoryID equals c.CategoryID
    join s in dcNwind.Suppliers on p.SupplierID equals s.SupplierID
    select new XElement("Order_Detail",
        new XElement(ads + "ProductID", d.Element(ads + "ProductID").Value,
            new XAttribute(adsm + "type", "System.Int32")),
        <!-- ... -->
        new XElement(ads + "CategoryID", c.CategoryID.ToString(),
            new XAttribute(adsm + "type", "System.Int32")),
        new XElement(ads + "CategoryName",
            c.CategoryName),
        new XElement(ads + "SupplierID", s.SupplierID.ToString(),
            new XAttribute(adsm + "type", "System.Int32")),
        new XElement(ads + "SupplierName", s.CompanyName),
        <!-- ... -->
        new XElement(ads + "Discount",
            d.Element(ads + "Discount").Value,
            new XAttribute(adsm + "type", "System.Single"))))))));
```

These examples also substitute the NorthwindDataContext's Product entity or the ProductList collection for the Products.xml InfoSet as the source of the ProductName element.

#### VB 9.0 Literal XML

```
Dim dcNwind As New NorthwindDataContext
<!-- ... -->
<Order_Details
<%= From d In xdDetails...<content> _
    Where d...<ads:OrderID>.Value = o...<ads:OrderID>.Value _
    Join p In dcNwind.Products _
        On p.ProductID.ToString() Equals d...<ads:ProductID>.Value _
    Join c In dcNwind.Categories On c.CategoryID Equals p.CategoryID _
    Join s In dcNwind.Suppliers On s.SupplierID Equals p.SupplierID _
    Select New XElement( _
        <Order_Detail>
```

## Chapter 7: Manipulating Documents with LINQ to XML

```
<ads:ProductID adsm:type="Int32">
    <%= d...<ads:ProductID>.Value %>
</ads:ProductID>
<!-- ... -->
<ads:CategoryID adsm:type="Int32">
    <%= c.CategoryID.ToString() %>
</ads:CategoryID>
<ads:CategoryName>
    <%= c.CategoryName %>
</ads:CategoryName>
<ads:SupplierID adsm:type="Int32">
    <%= s.SupplierID.ToString() %>
</ads:SupplierID>
<ads:SupplierName>
    <%= s.CompanyName %>
</ads:SupplierName>
<!-- ... -->
</Order_Detail> _ 
%>>
</Order_Details>
```

Figure 7-13 shows the HeterogeneousXmlJoinsVB.sln project displaying the output of the btnSupplCatJoinSQL\_Click event handler with the added elements highlighted. This screen capture also illustrates the duplicate namespace declarations.

The screenshot shows a Windows application window titled "Heterogeneous Joins and Lookups Between Documents, Objects, and Relational Tables (VB)". The main area displays an XML document with several namespace declarations. Notable highlights include the `ads:CategoryID`, `ads:CategoryName`, `ads:SupplierID`, `ads:SupplierName`, and `ads:ProductName` elements, which are underlined in yellow. The XML structure includes root elements like `<feed>`, `<entry>`, `<Order>`, `<Order_Detail>`, and nested elements such as `<ads:OrderID>`, `<ads:OrderDate>`, `<ads:RequiredDate>`, `<ads:ShippedDate>`, `<ads:Freight>`, `<ads:ShipName>`, `<ads:ShipAddress>`, `<ads:ShipCity>`, `<ads:ShipRegion>`, `<ads:ShipPostalCode>`, `<ads:ShipCountry>`, `<ads:Quantity>`, `<ads:UnitPrice>`, `<ads:Discount>`, and `<ads:CategoryID>`. At the bottom of the window, there are several tabs: "Order\_Details Lookup (XML)", "Products Join (XML)", "Products Join (SQL)", "Suppl./Cat. Join (SQL)", and "Take: 100". Below the tabs, there are checkboxes for "Display in IE", "Use Functional Construction", and performance metrics: "Load Time: 0.565 s.", "Query time: 4.897 s.", and "Characters: 316,339".

Figure 7-13

# Summary

LINQ to XML supplants many of the traditional XML document management, querying, and authoring tools provided by the CLR. The origins of LINQ include Microsoft Research's Xen and Cω languages, which were targeted at removing — or at least minimizing — XML/object (X/O) mismatch, so this chapter started with a brief introduction to the history of LINQ to XML and the features of .NET 3.5's `System.Linq.Xml` namespace. For most object types, LINQ is primarily a query language, so the chapter's examples began with elementary C# and VB query syntax over moderately complex hierarchical XML Infosets. VB supports a subset of XML IntelliSense features for composing queries when you infer a schema for the source document with VS 2008's improved XML Editor and XML schema inference engine. Adding the document and its schema to a project lets you take advantage of VB's XML axis properties to simplify query expressions.

Composing Infosets with C# 3.0's functional construction syntax is easier than using the CLR's `DOM` or `XmlTextWriter` tools. However, literal XML is a VB 9.0 data type that enables pasting a well-formed Infoset directly to the VB Editor window and replacing static values with embedded LINQ queries and inserted value expressions enclosed by `<%= ... %>` symbol pairs. Sample Windows form projects provide nontrivial examples of C# and VB LINQ to XML coding techniques.

The chapter's midpoint introduced grouping repeated elements and aggregating numeric values, which traditionally has required expertise with XPath, XQuery, XSLT or all three. This chapter is one of the few sources of examples for using LINQ to XML to generate subtotals and running totals. XML namespaces is another topic that deserves more thorough treatment with real-world documents, so ADO.NET Data Services' extensions to the Atom 1.0 publishing protocol provided examples of default (Atom) namespaces, as well as Microsoft's `ads` and `adsm` namespace prefixes for the special purpose elements to support delivering data from Web services in the cloud. Supporting IntelliSense for VB queries over Infosets with namespaces requires downloading and installing the XML To Schema Wizard.

LINQ to XML works with all XML prefixes expanded, which hampers document readability and greatly increases their size. Therefore, the chapter described techniques for removing unneeded local namespace declarations from Infosets by string manipulation and document transformation methods. The chapter ended with detailed examples of code for adding element groups with lookups and inserting elements with LINQ to XML joins with documents, LINQ to SQL entities, and generic collections.

# 8

## Exploring Third-Party and Emerging LINQ Implementations

The C# 3.0 team designed LINQ to be extensible so Microsoft and third-party developers could write domain-specific LINQ derivatives. When Microsoft launches an intriguing new language feature, such as LINQ, the release unleashes evangelism by internal groups and rapid exploitation by independent software developers. Visual Studio 2008's LINQ to SQL, LINQ to XML and LINQ to DataSet, and ADO.NET's LINQ to Entities were Microsoft's only RTM domain-specific LINQ products when this book was written. However, the following four LINQ implementation projects were in the development stage at Microsoft in mid-2008:

- ❑ **Parallel LINQ** (also called PLINQ) enables you to take advantage of parallel process of LINQ queries by multiple-core processors. PLINQ is part of the Microsoft Parallel Extensions to .NET Framework 3.5, which the Parallel Fx team released as a Community Technical Preview (CTP) on December 5, 2007. The latest version was the June 2008 CTP when this book was written.
- ❑ **LINQ to REST** is an implementation for translating LINQ queries into Representational State Transfer URLs that define a request to a Web service. LINQ to REST debuted as a component of Web-based ADO.NET Data Services (formerly codenamed "Project Astoria") in a December 9, 2007 CTP. .NET 3.5 SP1 includes the release first release version.
- ❑ **LINQ to XSD** is an incubator project to update the LINQ version from VS 2008 Beta 1 to VS 2008 RTM; Chapter 1's "LINQ to XSD" topic introduces you to the Alpha 0.2 release of this implementation.
- ❑ **LINQ to Stored XML** for SQL Server's `xml` data type is intended to supplement SQL Server 2005 and later's XQuery-based language. LINQ to Stored XML is another incubator project announced by Shyam Pathir at the XML 2007 Conference. No early version of LINQ to Stored XML was available when this book was written.

## Part III: Applying Domain-Specific LINQ Implementations

---

This chapter contains detailed descriptions of Parallel LINQ and LINQ to REST, as well as sample project files that demonstrate capabilities of the versions that were available when this book was written.

Third-party LINQ implementers range from professional developers adding full-featured LINQ query capability to commercial software to enthusiasts writing simple mashups of existing Web applications. Following are the two more sophisticated third-party implementations that this chapter describes:

- ❑ **LINQ to Active Directory** (also called LINQ to AD and formerly LINQ to LDAP) by Bart de Smet enables querying ActiveDirectory domains.
- ❑ **LINQ to SharePoint**, also by Bart de Smet, brings query capability to SharePoint lists with Collaborative Application Markup Language (CAML).

Four of the preceding six implementations share a common characteristic: They use expression trees to translate LINQ expressions to REST URLs, LDAP (Lightweight Directory Access Protocol), and CAML. The source code for LINQ to Active Directory and LINQ to SharePoint is publicly available on CodePlex ([www.codeplex.com/LINQtoAD](http://www.codeplex.com/LINQtoAD) and [www.codeplex.com/LINQtoSharePoint](http://www.codeplex.com/LINQtoSharePoint)).

*The last update to LINQ to SharePoint prior to this book's publication occurred on November 30, 2007. LINQ4SP is a similar, commercial SharePoint implementation by L&M Solutions of Hungary. You can learn more about LINQ4SP at <http://lmsolutions.web.officelive.com/linq4sp.aspx>.*

## Emerging Microsoft LINQ Implementations

It's a good bet that LINQ and the language extensions to support it aroused more developer interest than the C#, VB, and Visual Studio teams expected. Therefore, the Developer Division sought to leverage their initial four LINQ implementations by initiating in late 2007 formal development programs for the four domain-specific LINQ implementations described in the following section.

### Parallel LINQ

Adding more cores to PC microprocessors has become the preferred approach to increasing CPU performance while minimizing incremental power consumption and thermal radiation. Prior to the advent of multi-core processors, increasing clock speed was the only method of improving performance of a specific CPU design. Decreasing the lithographic line width mitigates the increase in thermal radiation as clock speed increases at the expense of building new CPU fabrication (fab) lines and plants. Intel's new 45-nanometer (nm) process (code-named Penryn) supports up to 800 million transistors on a single chip. Intel's planning clock speeds of 3 GHz and greater for dual-core Penryn chips used in desktop PC desktops and servers, but going below 45 nm and above 3 GHz will be extremely costly compared with putting two Penryn chips in a package to create quad-core packages that share a 12-MB or larger Level 2 cache.

Operating systems and server applications, such as SQL Server 2000+, Windows Server 2003+, and others, are designed to take advantage of multiple CPUs, whether in individual packages or on a single die. Windows Vista accommodates quad-core CPUs but most client-side applications, written by in-house developers or independent software vendors, aren't designed to take full advantage of even dual-core CPUs. Sophisticated multi-threaded applications aren't easy to write or simple to test thoroughly.

## Chapter 8: Exploring Third-Party and Emerging LINQ Implementations

---

Parallel LINQ (PLINQ) is a component of the Microsoft Parallel Extensions to .NET Framework 3.5, which in turn is part of the Microsoft Parallel Computing Initiative (PCI). The object of the PCI is to take better advantage of multiple-core processors by taking the pain out of parallel (concurrent) processing of LINQ to Objects, LINQ to XML, and LINQ to DataSet queries. PLINQ is less applicable to LINQ to SQL and LINQ to Entities queries because queries are processed primarily by database servers. However, PLINQ can speed processing heterogeneous queries with sequences from relational domains. PLINQ parallelizes queries by relatively simple modifications to LINQ expression (comprehension) syntax and only slightly more cumbersome modifications of method call syntax. LINQ's declarative programming methodology, which describes the result that's wanted instead of how to accomplish the result, gives PLINQ the flexibility to accomplish its objectives without resorting to complex programming structures.

*The starting point for additional information on PLINQ and the Parallel Extensions is the Microsoft's Parallel Computing Center (<http://msdn.microsoft.com/en-us/concurrency/default.aspx>). The Parallel Extensions also contain the Task Parallel Library (TPL) and a hidden implementation of the Coordination Data Structures (CDS), which are beyond the scope of this chapter's declarative LINQ programming. The June 2008 CTP includes CDS documentation.*

### Programming with PLINQ

Using PLINQ with a project requires the following three operations:

1. Download and run the Microsoft Parallel Extensions to .NET Framework 3.5, June 2008 (or later) Community Technology Preview. (Search for "Microsoft Parallel Extensions" with the quotation marks.)
2. Add a reference to `System.Threading.dll` to add the `System.Threading` namespace to your project.
3. Add `using System.Threading;` or `Imports System.Threading` to your classes that contain LINQ queries.
4. Replace the Standard Query Operator's `IEnumerable<T>` type for sequences with `System.Linq.ParallelEnumerable<T>` by calling the `System.Linq.ParallelEnumerable.AsParallel<T>` extension method. `AsParallel` and `AsParallel<T>` appear in IntelliSense's AutoComplete list between `AsEnumerable<T>` and `AsQueryable`.

The `System.Threading` namespace extends `System.Linq` with the following two namespaces to implement PLINQ:

- ❑ `System.Linq` extends .NET 3.5's `System.Linq` namespace with `IParallelEnumerable<T>` and `IParallelOrderedEnumerable<T>` interfaces and `ParallelEnumerable`, `ParallelQuery` and `ParallelQueryOptions` classes. `ParallelEnumerable` provides the SQO parallel processing extension methods. `ParallelQuery` provides the `AsParallel()`, `AsParallel<T>()`, and `AsSequential<T>()` methods. `ParallelQueryOptions` supports the `PreserveOrdering` option to force parallel processing of the output sequence to following the input sequence and the integer `DegreeOfParallelism` argument that specifies the maximum number of threads. `System.Linq` also includes a number of related internal sealed / Friend NotInheritable classes.
- ❑ `System.Linq.Parallel` adds the `Parallel` namespace with an `Enumerable` class of extension methods and a large number of internal sealed / Friend NotInheritable helper classes.

## Part III: Applying Domain-Specific LINQ Implementations

---

Add the \Program Files\Microsoft Parallel Extensions Jun08 CTP\System.Threading.dll file to Lutz Roeder's Reflector application if you want to explore the classes in detail. Note that Jun08 might have changed to a later date.

The `IParallelEnumerable<T>` interface derives from `IEnumerable<T>` and adds only the overhead needed to enable binding to PLINQ's `ParallelEnumerable` query provider. Here's the basic syntax for LINQ expression and method call syntax with the `PreserveOrdering` option and four threads maximum specified:

### C# 3.0

```
// Sequential (standard) query - method call syntax (C#)
var query = dataSource
    [.Where(a => a.Prop1Name == someValue)
     .orderby(a => a.Prop2Name)
     .Select(a)];

foreach (var a in query)
{
    // Do something with a
}

// Parallel (pipelined) query - method call syntax
var query = dataSource
    .AsParallel([QueryOptions.PreserveOrdering[, 4]])
    [.Where(a => a.Prop1Name == someValue)
     .orderby(a => a.Prop2Name)
     .Select(a)];

foreach (var a in query)
{
    // Do something with a
}

// Parallel (pipelined) query - query expression syntax
var query = for a in dataSource.AsParallel([QueryOptions.PreserveOrdering[, 4]])
    [where a.Prop1Name == someValue
     orderby a.Prop2Name]
    select a;

foreach (var a in query)
{
    // Do something with a
}
```

### VB 9.0

```
' Sequential (standard) query - method call syntax (VB)
Dim Query = DataSource _
    [.Where(Function(a) a.Prop1Name = someValue) _
     .orderby(Function(a) a.Prop2Name) _
     .Select(a)]

For Each a in Query
```

```
' Do something with a
Next a

' Parallel (pipelined) query - method call syntax
Dim Query = DataSource _
    .AsParallel([QueryOptions.PreserveOrdering[, 4]]) _
    [.Where(Function(a) a.Prop1Name = someValue) _
     .OrderBy(Function(a) a.Prop2Name) _
     .Select(a)]

For Each a in Query
    ' Do something with a
Next a

' Parallel (pipelined) query - query expression syntax
Dim Query = For a In dataSource.AsParallel([QueryOptions.PreserveOrdering[, 4]]) _
    [Where a.Prop1Name = SomeValue _
     Order By a.Prop2Name] _
     Select a

For Each a in Query
    ' Do something with a
Next a
```

### Processing Queries

PLINQ offers the following three query processing options:

- ❑ **Pipelined processing** uses  $n - 1$  threads to run the parallel query and one thread to enumerate the query, where  $n$  is the number of processor cores in the machine. This method ensures that the `foreach/For Each...Next` loop can process elements as they exit parallel processing, which minimizes memory consumption for temporary storage of elements. When the enumerator thread invokes `MoveNext()` method, the processing thread(s) deliver a result as quickly as possible; the enumerator thread blocks until a result is available. Pipelined processing is the default unless you invoke the `ToArray()`, `ToDictionary()`, `ToList()`, or `ToLookup()` method or include the `OrderBy()` or `OrderByDescending()` SQO in your query. Pipelined processing uses the basic query syntax shown in the preceding examples.
- ❑ **Stop-and-go processing** starts with a call to `MoveNext()`. The enumerator thread then enlists threads from the remaining cores to complete parallel query processing. This method stores elements in memory; the first thread enumerates the output and the subsequent `MoveNext()` invocation when parallel query processing completes returns the entire output. Additional `MoveNext()` invocations return the same data from the buffer. You select stop-and-go processing by invoking PLINQ's `GetEnumerator(pipelined)` overload with `pipelined = false`. Stop-and-go processing is used for `To...()` methods and queries sorted with `OrderBy[Descending]()` because these operators force all-at-once processing. Stop-and-go processing uses the iteration syntax shown in the following example.
- ❑ **Inverted enumeration** requires supplying a lambda function to run in parallel one time for each element. Inverted enumeration doesn't incur the overhead of merging the output from multiple threads. Inverted enumeration requires substituting the `ForAll` API for `foreach/For Each...Next` and the lambdas cannot share state. The later "Ray Tracing" section has an example of inverted enumeration with the `ForAll` API.

## Part III: Applying Domain-Specific LINQ Implementations

---

### C# 3.0

```
// Enumeration with stop-and-go processing (C#)
using (var q = query.GetEnumerator(false))
{
    while (q.MoveNext())
    {
        // the first invocation starts query processing
        // then enumerate the results with
        operationOn(q.Current);
    }
}
```

### VB 9.0

```
' Enumeration with stop-and-go processing (VB)
Using q = Query.GetEnumerator(False)
    While q.MoveNext()
        ' the first invocation starts query processing
        ' then enumerate the results with
        OperationOn(q.Current);
    End While
End Using
```

## Running the PLINQ Samples

Installing the Microsoft Parallel Extensions to .NET Framework 3.5, June 2008 Community Technology Preview adds a Microsoft Parallel Extensions Jun08 CTP node to the Start menu that has Docs and Samples folders. Choosing Samples opens the ... \Samples folder that has subfolders for individual examples, as well as multiple PLINQ and TPL projects. The LINQRayTracer example is a C# project that uses LINQ has C#, F#, and VB.

### LINQRayTracer

Ray tracing is a computationally intensive process for photorealistic image rendering. According to Wikipedia:

Ray tracing is a . . . rendering algorithmic approach in 3D computer graphics, where mathematically-modeled visualizations of programmed scenes are produced using a technique which follows rays from the eyepoint outward, rather than originating at the light sources. It produces results similar to ray casting and scanline rendering, but facilitates more advanced optical effects, such as accurate simulations of reflection and refraction, and is still efficient enough to frequently be of practical use when such high quality output is sought.

The LINQRayTracer project is based on an original C# project by Luke Hoban, who's now a program manager for the F# team. Hoban ported the C# code to version 3.0 and then to a sequential LINQ to Objects query, as described in his <http://blogs.msdn.com/lukeh/archive/2007/04/03/a-ray-tracer-in-c-3-0.aspx> post of April 3, 2007. Figure 8-1 illustrates the program after parallel-rendering the image, which required 23.73 seconds; sequential rendering required 33.30 seconds on a 2.80 GHz dual-core Pentium CPU running Windows Vista Ultimate on a Gateway S-5200D desktop PC with 4.0 GB.

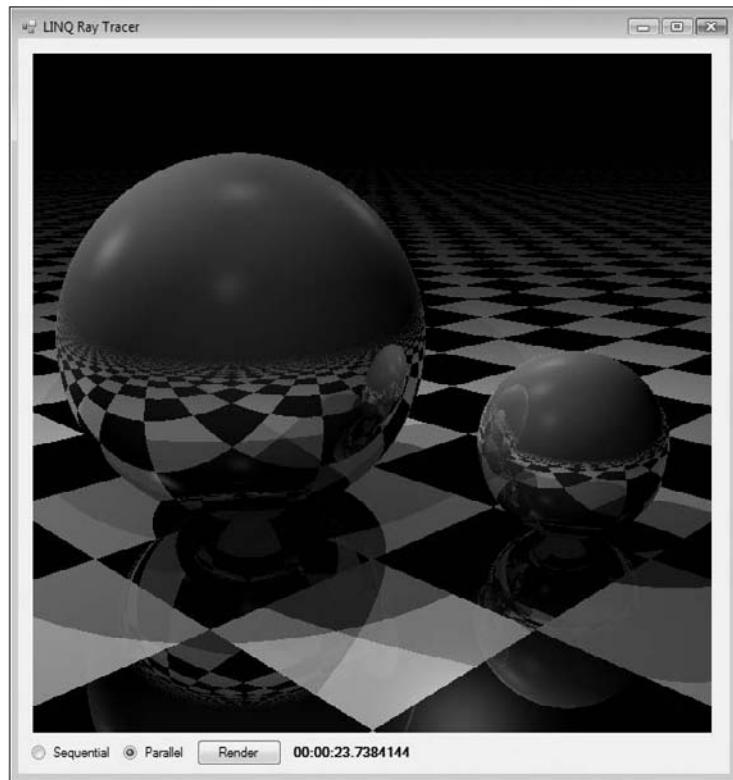


Figure 8-1

The `RenderParallel()` method's PLINQ-enabled LINQ query starts with

### C# 3.0

```
var pixelsQuery =
    from y in ParallelEnumerable.Range(0, screenHeight)
    let recenterY = -(y - (screenHeight / 2.0)) / (2.0 * screenHeight)
    select from x in Enumerable.Range(0, screenWidth)
    ...
```

and continues with 155 more lines, which probably qualified it as the world's most complex LINQ query when Hoban authored it in March 2007. There are 283 additional lines in `Scene`, `Surface`, `Vector`, `Color`, `Ray`, `Camera`, `Light`, `Sphere`, and `Place` classes of the `LINQRayTracer.cs` file, not including the `RenderParallel()` method.

## Part III: Applying Domain-Specific LINQ Implementations

---

The enumerator code uses the `ForAll` API with a `row` lambda expression to specify inverted enumeration processing:

### C# 3.0

```
int rowsProcessed = 0;
pixelsQuery.ForAll(row =>
{
    foreach (var pixel in row)
    {
        rgb[pixel.X + (pixel.Y * screenWidth)] = pixel.Color.ToInt32();
    }
    int processed = Interlocked.Increment(ref rowsProcessed);
    if (processed % rowsPerUpdate == 0 ||
        processed >= screenHeight) updateImageHandler(rgb);
});
});
```

### VB and C# Samples

The VB and C# samples consist of the following individual algorithms for generating and testing numeric values, strings, or geometric shapes executed in sequence:

1. **RectangleUnion** computes the area of the union of N rectangles with sides parallel to x and y axes in  $O(N^2)$  time.
2. **CountValidISBNs** counts how many strings in the input correspond to valid ISBN numbers (with correct checksum values).
3. **NQueens** counts how many ways there are to place N queens on an  $N \times N$  chessboard so that no two queens attack each other.
4. **WordScrambler** returns a new array of all 188,462 words from Charles Dickens' *Great Expectations* with all but the first and last letters scrambled.

The following table summarizes performance data generated by code added to the sample classes for the five preceding algorithms.

Test Algorithm	Parallel Time, ms.	Sequential Time, ms.	Ratio: Par./Seq.
RectangleUnion	366	665	0.5504
CountValidISBNs	117	217	0.5392
NQueens	204	372	0.5484
WordScrambler	659	1013	0.6505

Figure 8-2 shows task manager's trace captured during execution of the four test algorithms. Notice that overall usage of the second (right) core is slightly greater than the first (left) core.

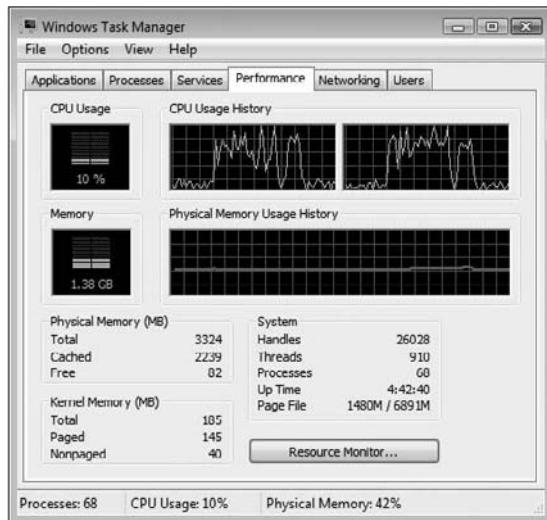


Figure 8-2

It's clear from the preceding test data with a low-end, two-core processor based on a c. 2006 CPU that PLINQ delivers a significant performance boost to the four algorithms. It's likely that one of today's quad-core processors designed for use in desktop PCs and low-end servers, together with future tuning of the PLINQ code, would deliver even greater performance gains.

*You can follow Microsoft's PCI development program from links at the Parallel Computing Developer Center (<http://msdn2.microsoft.com/en-us/concurrency/default.aspx>) and the Parallel Programming with .NET blog (<http://blogs.msdn.com/pfxteam/>).*

## LINQ to REST

Roy Fielding, one of the authors of the Internet Engineering Task Force's HTTP specification (RFC 2216), introduced the term *Representational State Transfer* and its abbreviation *REST* in his 2000 doctoral dissertation, "Architectural Styles and the Design of Network-based Software Architectures." Wikipedia's definition of REST begins as follows:

REST strictly refers to a collection of network architecture principles that outline how resources are defined and addressed. The term is often used in a looser sense to describe any simple interface that transmits domain-specific data over HTTP without an additional messaging layer such as SOAP or session tracking via HTTP cookies. . . . Systems that follow Fielding's REST principles are often referred to as *RESTful*.

## Part III: Applying Domain-Specific LINQ Implementations

---

LINQ to REST is a component of ADO.NET Data Services (ANDS), which began life in early 2007 as a Microsoft Live Labs incubator project named *Astoria*. Astoria used the term REST in the “looser sense” to describe an API that defines a method for delivering relational data over the Web with a simpler API than SOAP. At that time, many Web-centric organizations were creating “open,” RESTful APIs for accessing a variety of data types, including data structures, that support HTTP as the wire protocol and integrate easily with AJAX and other technologies, such as Adobe Flash and Microsoft Silverlight, to deliver lightweight client programs that run in popular browsers. The common term for these programs, which developers often use to create Web *mashups*, is Rich Internet Application (RIA).

*The incubation team chose the codename Astoria because the project originally was subtitled “Data Service in the Cloud” and Astoria is the cloudiest city in the United States.*

*Mashup originated as a pop-music term that means creating a new tune by mixing multiple existing tunes. In the Internet context, mashup means combining data from multiple sources into a Web page or application. An early example was associating Web-based maps with real-estate classified advertisements on Craigslist and similar sites.*

*Microsoft insists that RIA is the abbreviation for Rich Interactive Applications.*

REST transfers state between resources (a source of specific information) and clients. Resources, such as data tables, rows, and columns, must be uniquely addressable. RESTful resources must expose to clients a uniform interface that has a set of predefined operations and content types; these requirements are similar to those of a SOAP service’s contract. ANDS meets these criteria. The client/server protocol for transferring state must be stateless, cacheable and layered. HTTP satisfies these requirements, enables state to pass through most firewalls, and provides predefined methods for client authentication.

Google Base, an attribute-centric (non-relational) database, was an early RESTful, publicly accessible Web data source; Amazon announced in December 2007 the beta version of SimpleDB, another cloud-based non-relational RESTful data store that stores data in key-value pairs. The initial Astoria implementation, which featured a publicly accessible Web implementation of the Northwind sample database and a few other data sources, used the Entity Framework (EF) as its preferred back-end object/relational data store. ANDS with Entity Framework as object data source is one of the topics of Chapter 15. Figure 8-3 is a data flow diagram for ADO.NET Data Services using EF, LINQ to SQL, or other LINQ-compliant data sources.

## Chapter 8: Exploring Third-Party and Emerging LINQ Implementations

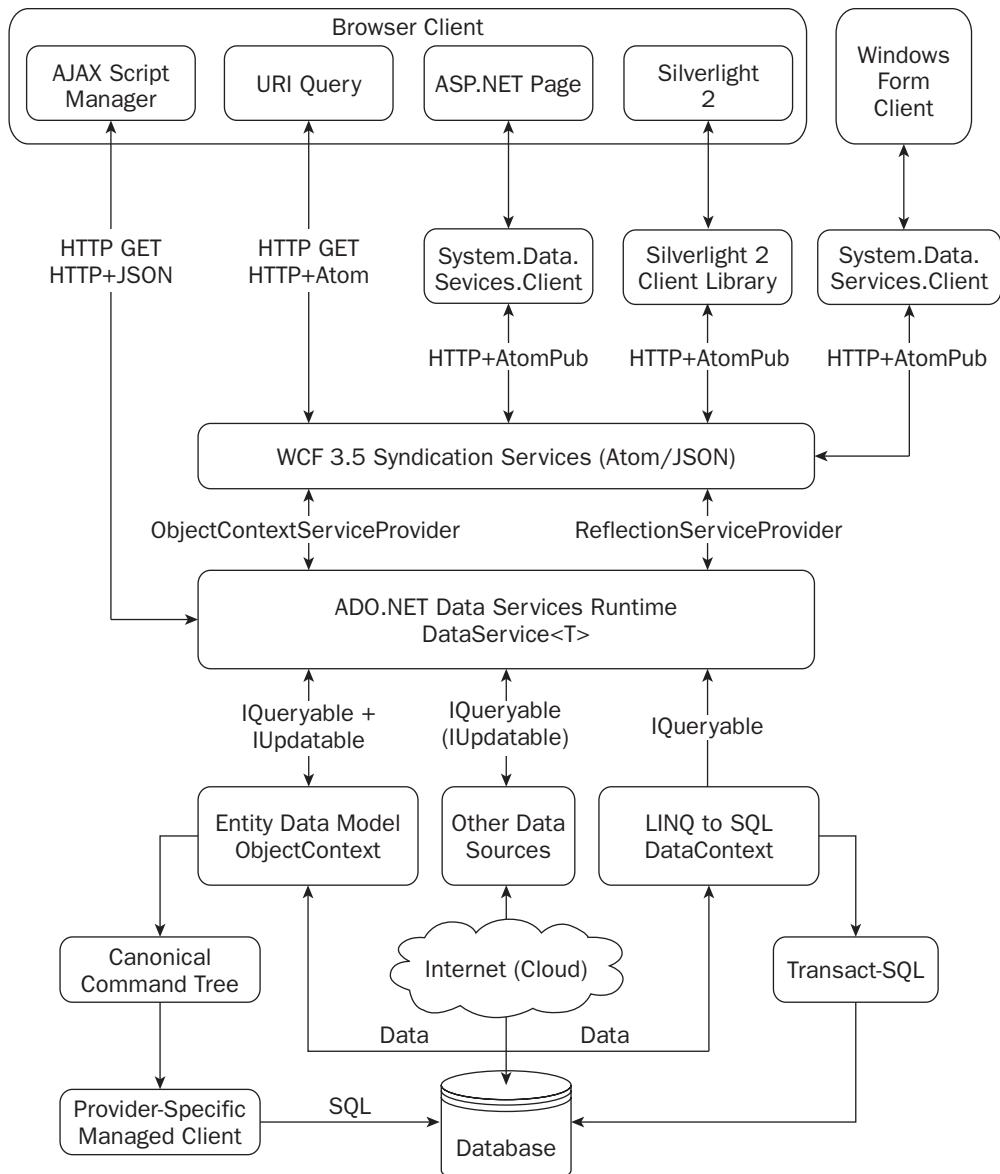


Figure 8-3

## Part III: Applying Domain-Specific LINQ Implementations

---

Astoria implements the `IUpdatable<T>` interface for EF so EF data sources are updatable. LINQ to SQL and other data sources that don't implement `IUpdatable<T>` are read-only.

### Adopting URIs as a Query Language

Universal Resource Identifiers (URIs) replace SQL statements as the foundation for ANDS' query language, which is based on EF's Entity Data Model (EDM). EDM introduces three-layer object/relational mapping between a conceptual schema (CSDL), which defines the domain's business objects and a physical schema (SSDL), which represents the underlying relational data store's schema. A middle layer, called the mapping schema (MSL), defines the relationship between the conceptual and physical schemas. All three schemas are complex XML files stored in your VS 2008 project's executable (...\\bin) folder: `ModelName.csdl`, `ModelName.msl`, and `ModelName.ssdl`. EF has its own Entity SQL (eSQL) query language, which is an SQL syntax with extensions to support queries against entity collections related by associations that the Entity Services layer exposes (refer to Figure 8-3). Although ANDS' URI query syntax is based on the EDM, ANDS v1.0 will support back-end data sources other than EF. The primary requirement is that the data source must support the `IQueryable<T>` interface and a new interface, `IUpdatable<T>`, for read-write data.

ANDS' URIs follow the REST policy of emphasizing nouns to identify resources as opposed to SOAP services that use verbs to invoke service methods. ANDS resources are Windows Communication Foundation (WCF) services that use the server name, following by the EDM's `EntityContainer` name, which is same as the `ConnectionString` name by default, with a .svc suffix as the URL component of the URI, as in the following HTTP GET request:

```
http://localhost[:portNumber]/northwind.svc/
```

for a local resource based on the Northwind sample database. TCP `portNumber` is required only if the service doesn't use well-known HTTP port 80 or HTTPS port 443.

Executing this URL in IE returns a list of the resource's `Entities` (collections) in an extended version of Atom 1.0 format (see Figure 8-4) and the Atom Publishing Protocol (APP):

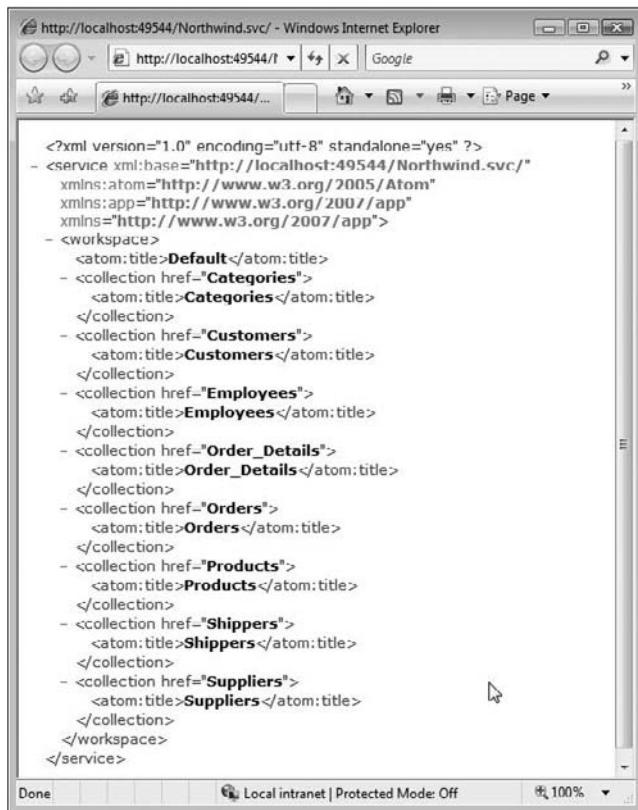


Figure 8-4

Chapter 7's "Working with XML Namespaces and Local Names" section describes the Atom 1.0 syndication format. ANDS also supports the JavaScript Object Notation (JSON) format for JavaScript object compatibility. JSON examples are beyond the scope of this chapter.

### Running the Sample Northwind.svc WCF Service and Client Projects

The \WROX\ADONET\Chapter08\CS\NwindDataServiceCS\NwindDataServiceCS.sln and . . . \VB\ NwindDataServiceVB\NwindDataServiceVB.sln projects let you experiment with ANDS URI query syntax with VS 2008 SP1, which includes RTM versions of ADO.NET Data Services v1 and EV v1.

### Experimenting with URI Query Syntax

Navigate to and double-click . . . \ NwindDataServiceCS.sln or NwindDataServiceVB.sln to start the service and open IE. Add /northwind.svc/ to the URL to display the Northwind.svc service's collections in Atom 1.0 syndication format (refer to Figure 8-4).

*If you see a page with a "You are viewing a feed that contains frequently updated content" or an "Internet Explorer does not support this feed format" message, you must disable formatting of Atom feeds in IE 7+ by opening the Internet Options dialog, clicking the Content tab and Settings button, and clearing the Turn on Feed Reading View check box. Check OK twice to close the dialog, then close IE 7+. Press F5 to restart the service.*

## Part III: Applying Domain-Specific LINQ Implementations

To access a specific `EntitySet`, add the Entity name, such as the following for a list of Products:

```
http://localhost:52660/northwind.svc/Products/
```

as shown in Figure 8-5, which displays the first Atom `<feed><entry>` group.

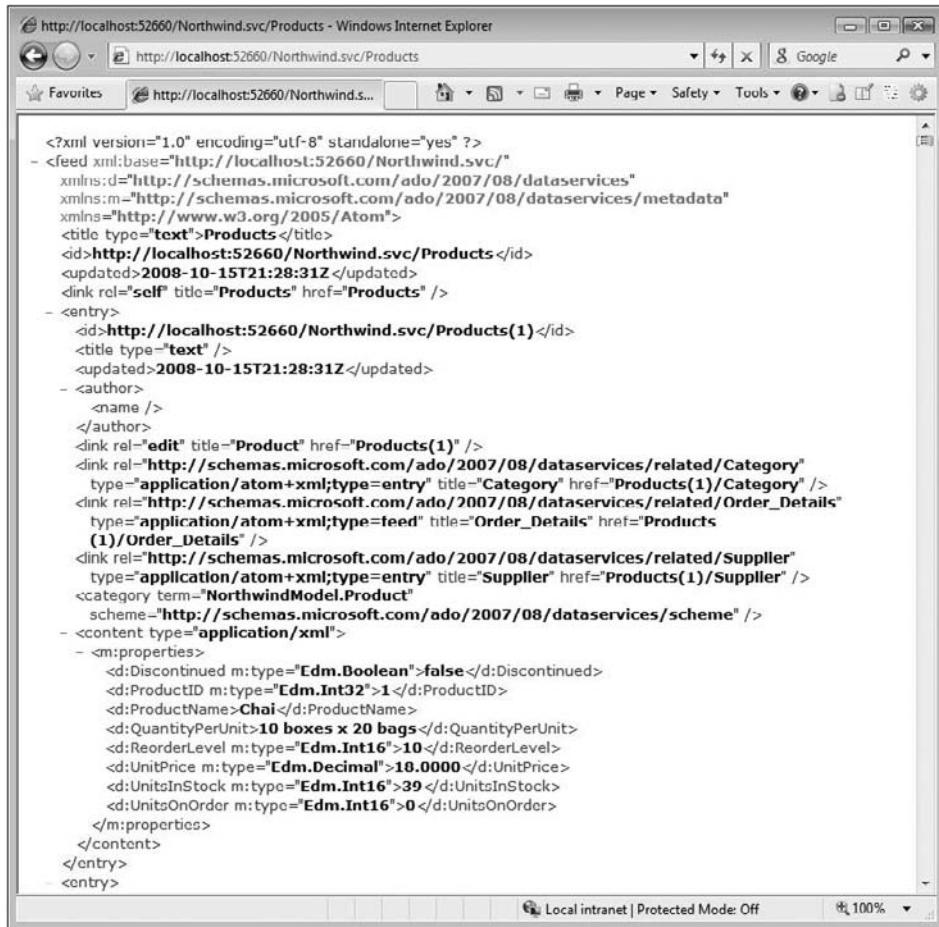


Figure 8-5

*Parts of the URI to the right of the service name are case-sensitive for both C# and VB projects, regardless of whether the data store collation is case-sensitive or case-insensitive.*

The formal syntax for navigating ANDS resources is:

```
http://host/[vdir/]service/[EntitySet[(Key)] [NavigationProperty[(Key) / . . .]]]
```

## Chapter 8: Exploring Third-Party and Emerging LINQ Implementations

To access a specific Entity instance within the `EntitySet`, add a valid `Key` value in parentheses, such as:

```
http://localhost:52660/northwind.svc/Products(5)
```

to return the `Product` entity for Chef Anton's Gumbo Mix as an Atom 1.0 `<entry>` fragment without the root `<feed>` element (see Figure 8-6).

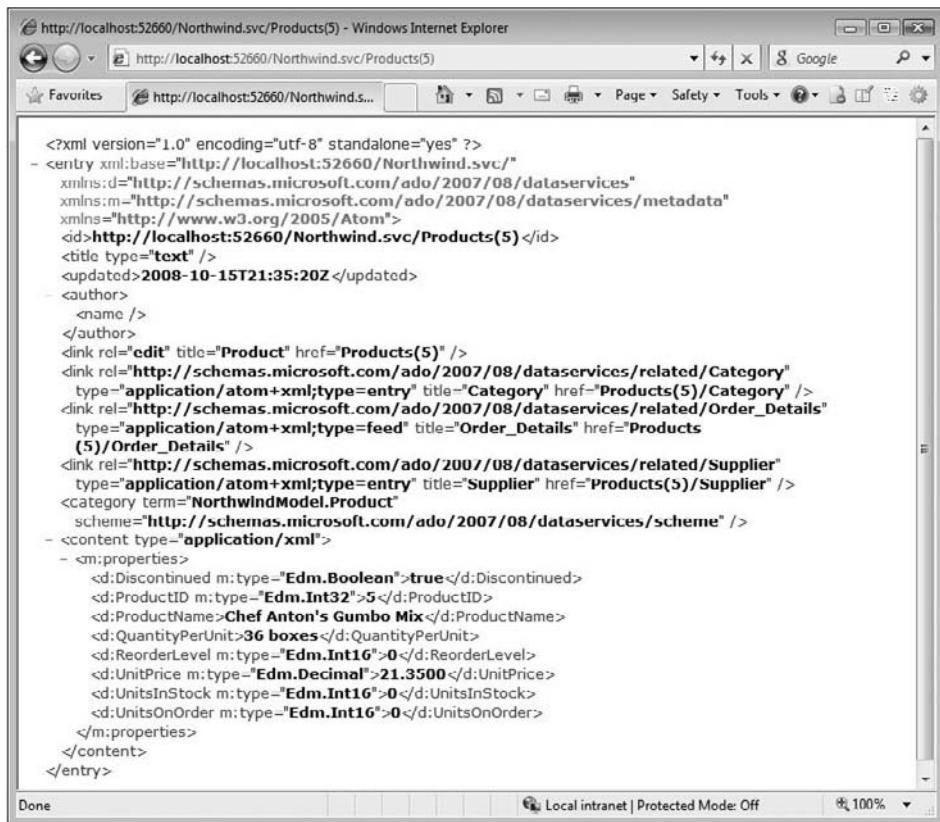


Figure 8-6

You also can take advantage of many:1 navigation properties, such as the `Product.Category` property by adding `/Category`, as in:

```
http://localhost:52660/northwind.svc/Products(5)/Category
```

## Part III: Applying Domain-Specific LINQ Implementations

---

which returns the following fragment for the Condiments category:

### XML (Atom 1.0)

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<entry xml:base="http://localhost:52660/Northwind.svc/"
    xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns="http://www.w3.org/2005/Atom">
    <id>http://localhost:52660/Northwind.svc/Categories(2)</id>
    <title type="text" />
    <updated>2008-10-15T21:38:00Z</updated>
    <author>
        <name />
    </author>
    <link rel="edit" title="Category" href="Categories(2)" />
    <link rel=
        "http://schemas.microsoft.com/ado/2007/08/dataservices/related/Products"
        type="application/atom+xml;type=feed" title="Products"
        href="Categories(2)/Products" />
    <category term="NorthwindModel.Category"
        scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    <content type="application/xml">
        <m:properties>
            <d:CategoryID m:type="Edm.Int32">2</d:CategoryID>
            <d:CategoryName>Condiments</d:CategoryName>
            <d>Description>
                Sweet and savory sauces, relishes, spreads, and seasonings
            </d>Description>
            <d:Picture m:type="Edm.Binary">
                FRwvAAIAAAANAA4AFAAhAP////
            </d:Picture>
        </m:properties>
    </content>
</entry>
```

Similarly, the following URL returns a `<feed>` document containing `<entry>` groups for each Product in Category 2 (Condiments):

[http://localhost:52660/Northwind.svc/Categories\(2\)/Products](http://localhost:52660/Northwind.svc/Categories(2)/Products)

ANDS' URI query strings have options, operators, and functions to provide a reasonably complete query syntax that supports hierarchies based on entity associations. The following three sections describe ANDS query options, operators and functions, as well as provide syntax examples for options and operators.

## Chapter 8: Exploring Third-Party and Emerging LINQ Implementations

### Query Options

The following table lists the five ANDS query options. Ellipsis (...) represents `http://serverName:tcpPort/serviceName.svc`, `http://localhost:52660/Northwind.svc` for this chapter's examples.

ADO.NET Data Services Query Options		
Option	Description	Example
expand	Includes groups for related entities inline with the target entity	<code>.../Customers('ALFKI')?\$expand=Orders</code> returns a <feed> document with Customer data and all Orders placed by the Customer  <code>.../Orders(10248)?\$expand=Employees,Shippers</code> returns an <entry> fragment with Order data and Employee, and Shipper groups
filter	Filters the entities returned by the <i>Property operator Criterion</i> expression (see Table 8-3)	<code>.../Orders?\$filter=ShipCity eq 'Paris'</code> returns all Orders shipped to Paris  <code>.../Orders?\$filter=ShipCity eq 'Paris' and OrderDate gt datetime'1997-12-31T00:00:00'</code> returns all Orders shipped to Paris after December 31, 1997
orderby	Sorts the group by the specific property value ascending ( <code>asc</code> , default) or descending ( <code>desc</code> )	<code>.../Orders?\$orderby=ShipCity</code> sorts alphabetically by city name  <code>.../Orders?\$orderby=ShipCity desc</code> reverses the sort order  <code>.../Orders?\$orderby=ShipCity desc,ShipName</code> adds ship name as a criterion
skip, top	Skips the number of instances specified by the <code>skip</code> parameter then takes the number of instances specified by the <code>top</code> parameter	<code>.../Orders?\$skip=5</code> returns all Orders after the first five  <code>.../Orders?\$skip=5&amp;\$top=2</code> returns two Orders after the first five  <code>.../Orders?\$skip=40&amp;\$top=10</code> returns the fifth page of 10 orders

### Logical, Arithmetic and Grouping Query Operators

The following table lists the ANDS logical, arithmetic, and grouping query operators.

## Part III: Applying Domain-Specific LINQ Implementations

ADO.NET Data Services Logical, Arithmetic and Grouping Query Operators		
Operator	Description	Example
Logical Operators		
eq	Equal	.../Orders?\$filter=ShipCity <b>eq</b> 'Paris'
ne	Not equal	.../Orders?\$filter=ShipCity <b>ne</b> 'Paris'
gt	Greater than	.../Orders?\$filter=UnitPrice <b>gt</b> 20
ge	Greater than or equal	.../Orders?\$filter=Freight <b>ge</b> 50
lt	Less than	.../Orders?\$filter=Freight <b>lt</b> 10
le	Less than or equal	.../Products?\$filter=UnitPrice <b>le</b> 50
and	Logical and	.../Products?\$filter=UnitPrice <b>le</b> 50 <b>and</b> UnitPrice gt 10
or	Logical or	.../Products?\$filter=UnitPrice <b>le</b> 50 <b>or</b> UnitPrice gt 10
not	Logical negation	.../Customers?\$filter= <b>not</b> endsWith(PostalCode, '50')
Arithmetic Operators		
add	Addition	.../Products?\$filter=UnitPrice <b>add</b> 5 <b>gt</b> 10
sub	Subtraction	.../Products?\$filter=UnitPrice <b>sub</b> 5 <b>gt</b> 10
mul	Multiplication	.../Orders?\$filter=Freight <b>mul</b> 800 <b>gt</b> 2000
div	Division	.../Orders?\$filter=Freight <b>div</b> 10 <b>eq</b> 4
mod	Modulo	.../Orders?\$filter=Freight <b>mod</b> 10 <b>eq</b> 0
Grouping Operators		
( )	Precedence grouping	.../Products?\$filter=(UnitPrice <b>sub</b> 5) <b>gt</b> 10

*Internet Explorer replaces spaces surrounding the operator with %20 when it URL-encodes the query string.*

## Chapter 8: Exploring Third-Party and Emerging LINQ Implementations

### String, Date, Math, and Type Query Functions

The following table lists the ANDS string, date, math, and type query functions with C# syntax.

ADO.NET Data Services String, Date Math and Type Query Functions	
String Functions	Date Functions
<code>string concat(string s0, string s1)</code>	<code>int day(DateTime d0)</code>
<code>bool contains(string s0, string s1)</code>	<code>int hour(DateTime d0)</code>
<code>bool endswith(string s0, string p1)</code>	<code>int minute(DateTime d0)</code>
<code>bool startswith(string s0, string s1)</code>	<code>int month(DateTime d0)</code>
<code>int length(string s0)</code>	<code>int second(DateTime d0)</code>
<code>int indexof(string arg)</code>	<code>int year(DateTime d0)</code>
<code>string insert(string s0, int pos, string s1)</code>	<b>Math Functions</b>
<code>string remove(string s0, int pos)</code>	<code>double round(double n0)</code>
<code>string remove(string s0, int pos, int length)</code>	<code>decimal round(decimal n0)</code>
<code>string replace(string s0, string find, string replace)</code>	<code>double floor(double n0)</code>
<code>string substring(string s0, int pos)</code>	<code>decimal floor(decimal n0)</code>
<code>string substring(string s0, int pos, int length)</code>	<code>double ceiling(double n0)</code>
<code>string tolower(string s0)</code>	<code>decimal ceiling(decimal n0)</code>
<code>string toupper(string s0)</code>	<b>Type Functions</b>
<code>string trim(string s0)</code>	<code>bool IsOf(type t0)</code>
	<code>bool IsOf(expression t0, type t1)</code>
	<code>&lt;t0&gt; Cast(type t0)</code>
	<code>&lt;t1&gt; Cast(expression t0, type t1)</code>

*URI queries have no aggregate functions in ADO.NET Data Services v1. The “Using ADO.NET Data Services” documentation states: “This release does not support Aggregate functions (sum, min, max, avg, etc.) as they would change the meaning of the ‘/’ operator to allow traversal through sets. For example, /Customers?\$filter=average(Orders/Amount) gt 50.00 is not supported.”*

## Part III: Applying Domain-Specific LINQ Implementations

---

### Processing Services Requests with the NwindServicesClient

ADO.NET Data Services provides a simple WCF HTTP client framework to support .NET 3.5 WinForms and WebForms clients, as well as a library for Silverlight 2 applications. The Windows and Web versions require a `using/Imports System.Data.Services.Client` directive that requires clients to reference the `System.Data.Services.Client.dll` library, which the ANDS setup program installs in the GAC. .NET clients you create with the library use the HTTP transport and Atom 1.0-formatted messages to communicate with an active (running) data service.

The `System.Data.Services.Client` namespace defines the following two classes:

- ❑ `ServiceDataContext` provides access to the runtime context of a data service, which you specify by the service URL: `http://serverName:tcpPort/serviceName.svc`. Like LINQ to SQL's `DataContext` and Entity Framework's `ObjectContext`, `ServiceDataContext` maintains state during successive interactions with the back-end `EntityContainer` to support updates to multiple entities and optional optimistic concurrency management.
- ❑ `ServiceDataQuery` objects represent URI queries against the underlying `EntityContainer` that return strongly typed .NET generic objects that you iterate inside familiar `foreach/For Each...Next` blocks.

Translating Atom 1.0 response messages into generic .NET objects or arrays of objects requires adding a service reference to the ADO.NET Data Service while it's running. To add the reference, right-click Solution Explorer's References node and select Add Service Reference to open the eponymous dialog. Type the URL for the service, `http://localhost:52660/Northwind.svc` for this example, in the Address text box, change the Namespace to Northwind for this example, and click OK to add the reference. You don't need to select an `EntitySet` in this case.

Alternatively, you can generate a class file for the client project by executing the `DataSvcUtil.exe` utility from the project's main source code folder (while the corresponding service is running) with the following command line:

```
"\Windows\Microsoft.NET\Framework\v3.5\DataSvcUtil.exe"  
/uri:http://localhost:52660/Northwind.svc  
/out:Northwind.cs [/language:CSharp]
```

*DataSvcUtil.exe generates a C# 3.0 class by default; you must add the /language:VB argument to return the VB 9.0 version.*

*The \WROX\ADONET\Chapter08\CS\NwindDSClientCS and . . . \VB\NwindDSClientVB folders contain sample NwindDSClientCS.sln and NwindDSClientVB.sln projects that operate with the data service described in the preceding sections. The client's default TCP port is 52660, which you can change in the TCP Port text box.*

The resulting class file contains a public partial class/`Partial Public Class` and a `ModelNameEntities` class, `NorthwindEntities` for this example, with default `DataServiceQuery` getters for each Entity.

# Chapter 8: Exploring Third-Party and Emerging LINQ Implementations

---

## Executing URI Queries

Following is the generic code for executing URI queries with the `DataServiceContext.CreateQuery<T>()` method, which takes the URI query string verbatim as its argument:

### C# 3.0

```
string url = "http://serverName:tcpPort/serviceName.svc";
Uri uri = new Uri(url);
DataServiceContext context = new DataServiceContext(uri);
context.MergeOption = MergeOption.AppendOnly;
DataServiceQuery<EntityType> query =
    context.CreateQuery<EntityType>("/EntityCollection[?$QueryOption(s)]");

foreach (EntityType e in query)
{
    // Do something with the e instance(s);
}
```

### VB 9.0

```
Dim url As String = http://ServerName: TcpPort/ServiceName.svc
Dim uri As Uri = New Uri(url)
Dim context As New DataServiceContext(uri)
context.MergeOption = MergeOption.AppendOnly
Dim Query As DataServiceQuery<EntityType> = _
    context.CreateQuery<EntityType>("/EntityCollection[?$QueryOption(s)]")

For Each e [As EntityType] In Query
    ' Do something with the e instance(s)
Next e
```

Clear the NwindDSClientCS.sln and NwindDSClientVB.sln projects' LINQ to REST Queries check box to execute sample URI query strings against the Products, Customers, or Orders EntityCollections, or emulate a GroupBy query.

## Executing LINQ to REST Queries

The `WebClient` API translates to URI queries simple LINQ expressions or method call lambda functions that specify the `EntityContainer.CreateQuery<T>()` method as the data source, as in the following generic example:

### C# 3.0

```
string url = "http://serverName:tcpPort/serviceName.svc";
ModelNameEntities entities = new ModelNameEntities(url);
entities.MergeOption = MergeOption.AppendOnly;
var query = from e in entities.CreateQuery<EntityType>("EntityCollection")
    where e.Property == value
    orderby e.Property [desc]
    select e;

foreach (EntityType e in query)
{
    // Do something with the e instance(s);
}
```

## Part III: Applying Domain-Specific LINQ Implementations

---

### VB 9.0

```
Dim Url As String = "http://ServerName: TcpPort / ServiceName.svc"
Dim Entities As New ModelNameEntities(url)
Entities.MergeOption = MergeOption.AppendOnly
Dim query = From e In Entities.CreateQuery<EntityType>("EntityCollection") _
    Where e.Property = Value _
    Order By e.Property [desc] _
    Select e

For Each e [As EntityType] In Query
    ' Do something with the e instance(s)
Next e
```

If you add a break on the `foreach/For Each` line, you can hover the mouse over the `query` variable and display the URI query string in a Data Tip, as shown in Figure 8-7.



Figure 8-7

You can copy the query string (without the French braces) to the Clipboard and paste it into the address bar for testing translated URI query strings independently of client execution.

### Preventing HTTP 400 — Bad Request Errors and Lazy Loading Associated Entities

LINQ to REST lets you write expressions or chain method calls that generate invalid URI queries. The compiler tests your LINQ query syntax but not the translated URI query string's validity. For example, following is a sample LINQ to REST query that causes the NwindDSClientCS.sln project to generate a not-executable URI query string for the `Customers EntityCollection`:

```
var query = from c in entities.CreateQuery<Customer>("Customers")
    where c.Country == "USA" && c.Orders.Any(o => o.Freight > 50)
    orderby c.CompanyName
    select c;
```

Here's the equivalent method call:

```
entities.CreateQuery<Customer>("Customers")
    .Where(c => c.Country == "USA")
    .Where(c => c.Orders.Any(o => o.Freight > 500))
```

WebClient translates either of the preceding queries into the following URI query:

```
http://localhost:52660/Northwind.svc/Customers?
$filter=((Country)%20eq%20('USA'))%20and%20(Orders(Freight)%20gt%20(50))
&$orderby=CompanyName
```

## Chapter 8: Exploring Third-Party and Emerging LINQ Implementations

---

which appears as follows with %20 replaced by spaces for readability:

```
http://localhost:52660/Northwind.svc/Customers?
$filter=((Country) eq ('USA')) and (Orders(Freight) gt (50))
&$orderby=CompanyName
```

The problem with the preceding URI query string is that `Orders` is a `EntitySet` (collection), so it doesn't have a `Freight` property; URI queries don't have `Any()` or `All()` operators, which represent set traversal. Runtime exceptions also occur from using the `Join`, `GroupBy`, and many other SQOs in LINQ to REST queries.

The workaround for lack of an `All()` or `GroupBy()` counterpart is to execute an equivalent of the `Where(c => c.Orders.Any(o => o.Freight > 500))` in the `foreach/For Each` loop.

If you don't apply the `$expands=EntityRefOrSet` option to eager-load associated `EntityReferences` or `EntitySets`, you must lazy-load them with an `LoadProperty(object, associatedProperty)` expression in the loop as in the following example:

```
NorthwindEntities entities = new NorthwindEntities(url);
entities.MergeOption = MergeOption.AppendOnly;
var query = from c in entities.CreateQuery<Category>("Categories?$extend=Products")
            select c;

foreach (Category category in query)
{
    // Output category property values;
    // Lazy-load category.Products (f $extend=Products option isn't specified)
    // entities.LoadProperty(category, "Products");
    foreach (Product product in c.Products)
    {
        // Output product property values
    }
}
```

Figure 8-8 shows the NwindDSClientCS.sln project displaying the output of the preceding query with eager loading.

## Part III: Applying Domain-Specific LINQ Implementations

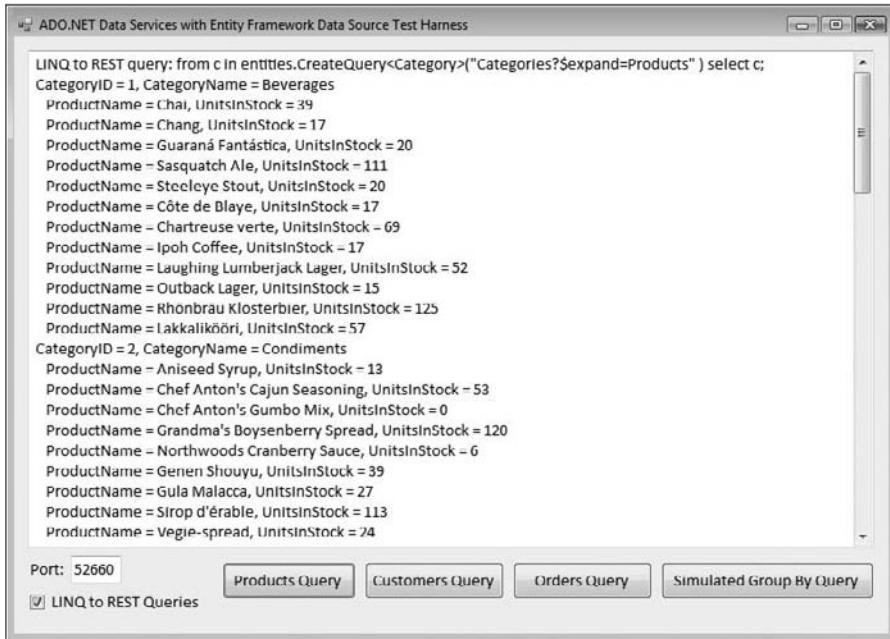


Figure 8-8

## LINQ to XSD

LINQ to XSD is a Microsoft Data Programmability group incubator project that supports strongly typed XML data sources for LINQ to XML queries. Instead of writing queries with element names as literal strings, as in the following query that totals the value of all items in a `purchaseOrder` document:

```
(from item in purchaseOrder.Elements("purchaseOrderItem")
    select (double)item.Element("Price") *
(int)item.Element("Quantity"))
.Sum();
```

You specify elements as typed objects (emphasized) in the corresponding LINQ to XSD query:

```
(from item in purchaseOrder.purchaseOrderItems
    select item.Price * item.Quantity)
.Sum();
```

LINQ to XSD reduces the programming mismatch between XML documents and business objects, so LINQ to XSD sometimes is known as *LINQ to XML Objects*. Obviously, the type-safe version of the query's syntax is clearer. It also provides IntelliSense for elements and corresponds more closely to a LINQ to SQL query against a `PurchaseOrder` entity with a `PurchaseOrderItems EntitySet` property.

LINQ to XSD maps XML Schemas (XSD) to object models, which enables XML data to be processed in typed manner in an object-oriented manner. Classes generated from the schema enforce validation constraints on elements and element groups in the same manner as the classes generated from strongly typed schemas that represent DataSets. LINQ to XSD enhances LINQ to XML by modeling typed views on untyped XML trees. Typed trees are available from most documents; if not, LINQ to XSD falls back to accessing untyped trees.

LINQ to XSD eliminates string-encoded access and casts; CLR namespaces replace XML namespaces. If you don't have a schema for the XML data you plan to convert to objects, you can create the required XSD schema with the XSD inference engine that's part of VS 2008's feature set. Installing LINQ to XSD previews or CTPs adds templates for LINQ to XSD Windows, Web, Console, and Library applications to the New Project dialog. After selecting the type of project to create, you add a new or existing XML Schema file to the project. Building the project generates the object model, enables IntelliSense for generated classes, and updates the Object Browser with the class information.

### **LINQ to XSD's History**

LINQ to XSD began life with availability of the first preview (Alpha 0.1) download, written by Ralf Lämmel of the Data Programmability (DP) Team, which became available for use with the May 2006 LINQ CTP on November 27, 2006. A few days later, Lämmel presented "Functional OO Programming with Triangular Circles," coauthored with Microsoft's Dave Remy, at the XML 2006 conference in early December 2006. The paper made a strong case for constructing and querying typed XML documents.

The team released a second preview (Alpha 0.2) for VS Orcas Beta 1 on June 8, 2007. This preview had very limited support for VB 9.0. In fact, code generation emitted only C# classes, which you had to add as a C# library to VB projects. Shortly thereafter, Lämmel left Microsoft to return to Germany as a professor of computer science at the new University of Koblenz (Germany).

On December 5, 2007 Microsoft's Shyam Pather gave a "LINQ to XML: Visual Studio 2008, Silverlight, and Beyond" presentation to the XML 2007 conference in Boston and discussed "incubation efforts for a post-Visual Studio 2008 technology, LINQ to XSD, which provides .NET developers with support for typed XML programming."

The DP team released a LINQ to XSD Preview Alpha 0.2 Refresh on February 20, 2008; search for *LINQ to XSD download* to and download `LinqToXsdSetup.msi`, which includes documentation and sample projects. The Refresh is compatible with VS 2008 RTM and SP1, and supports VB 9.0.

### **LINQ to Stored XML**

Shyam Pather also announced the LINQ to Stored XML incubation project at the XML 2007 conference. Hartmut Wilms reported the following in his "Post-VS 2008-Technology: LINQ to XSD and LINQ to Stored XML" article of December 5, 2007 for InfoQ.com ([www.infoq.com/news/2007/12/post-vs2008-linq-to-xml](http://www.infoq.com/news/2007/12/post-vs2008-linq-to-xml)):

LINQ to Stored XML (XML in the database) offers ways of issuing queries against XML data type columns within an SQL Server 2005. The goal is to "provide a strongly-typed LINQ experience over data in XML data type columns" by providing "mapping from XML schema to classes" and "query translation from LINQ

## Part III: Applying Domain-Specific LINQ Implementations

---

expressions to server XQuery expressions" (sample taken from the AdventureWorks database with 'Resume' being an XML data type column):

### Query [LINQ to Stored XML]

```
var q = from o in _data.JobCandidates
        where o.Resume.Skills.Contains("production")
        select o.Resume.Name.Name_Last;
```

### Output [SQL Server 2005+ XQuery expression]

```
SELECT [Extent1].[Resume].query(
    N'declare namespace r="http://..../adventure-works/Resume";
    /*[1]/r:Name/r:Name.Last'
    ).value(N'.', N'nvarchar(max)") AS [C1]
FROM [HumanResources].[JobCandidate] AS [Extent1]
WHERE cast(1 as bit) = ([Extent1].[Resume].query(
    N'declare namespace r="http://..../adventure-works/Resume";
    contains(/*[1]/r:Skills, "production")
    ).value(N'.', N'bit'))
```

LINQ to Stored XML uses LINQ to XSD features to strongly type elements in SQL Server 2005+ columns of the `xml` data type. It's clear from the preceding XQuery expression that LINQ to Stored XML will be a boon to developers who haven't mastered Microsoft's dialect of XQuery 1.0.

## Third-Party Domain-Specific LINQ Implementations

LINQ to Active Directory and LINQ to SharePoint were two of the early third-party LINQ implementations that were intended to demonstrate LINQ's capabilities when substituting `IQueryable<T>` for `IEnumerable<T>` and translating LINQ query syntax into languages other than T-SQL or LINQ to Entities' Canonical Query Trees (CQTs). Both of these applications are very useful because they output rather arcane expressions: Lightweight Directory Access Protocol (LDAP) filters and Collaborative Application Markup Language (CAML) for querying CAML objects, such as lists with SharePoint's Lists Web services.

### LINQ to Active Directory

LINQ to Active Directory, called *LINQ to AD* here for brevity, began life as LINQ to LDAP, a LINQ implementation by Belgian developer Bart De Smet that translated LINQ query expressions or method call chains into LDAP filters. The original project ran under VS Orcas Beta 1. Subsequently, De Smet joined Microsoft as a software design engineer (SDE) on the Windows Presentation Framework (WPF) team and updated LINQ to LDAP to LINQ to AD for the VS 2008 RTM version. You can download the LINQ to AD source code from [www.codeplex.com/LINQtoAD](http://www.codeplex.com/LINQtoAD).

A copy of the open-source project, LINQ to AD.sln, the BdsSoft.DirectoryServices.Linq library and its C# source code, and De Smet's original C# sample console project, Demo.csproj, are located in the `\WROX\ADONET\Chapter08\LINQtoAdSource` folder. The copyrighted source code in LINQtoAD.sln is licensed under a Microsoft Public License (Ms-PL), which is included with the project.

## Chapter 8: Exploring Third-Party and Emerging LINQ Implementations

According to the project's CodePlex home page, LINQ to ActiveDirectory offers the following features:

- Translates LINQ queries to LDAP filters according to RFC 2254, "The String Representation of LDAP Search Filters"
- Offers a simple entity model with support for updates
- Supports mappings to both the System.DirectoryServices (.NET) and ActiveDs (COM) APIs
- Ships with a set of samples

De Smet wrote a series of six blog posts named "The IQueryable Tales — LINQ to LDAP", which describe the development of LINQ to LDAP in detail. You can access the posts from

<http://community.bartdesmet.net/blogs/bart/archive/tags/LINQ/default.aspx>.

LINQ to AD's BdsSoft.DirectoryServices.Linq library includes references to Interop.ActiveDs to support mapping LDAP filters generated from LINQ queries to the Active Directory Services Interface (ADSI)'s ActiveDs.dll COM type library. Alternatively, a System.DirectoryServices reference supports LDAP mapping with fully managed code. The `DirectorySource.cs` class file contains the classes for defining the `DirectorySource<T>` generic type and generating and parsing the LINQ-to-LDAP expression trees.

The \WROX\ADONET\Chapter08\CS\LINQtoActiveDirectoryCS\LINQtoActiveDirectoryCS.sln and \WROX\ADONET\Chapter08\VB\LINQtoActiveDirectoryVB\LINQtoActiveDirectoryVB.sln projects are WinForm adaptations of the LINQtoAD.sln's Demo project. Both projects have a text box to enter the domain controller computer name for a test AD domain and include exception-handling code. By default, the sample project executes and displays the results of six sample LINQ to AD queries that code in the `MainForm.cs` or `MainForm.vb` event-handler defines (see Figure 8-9).

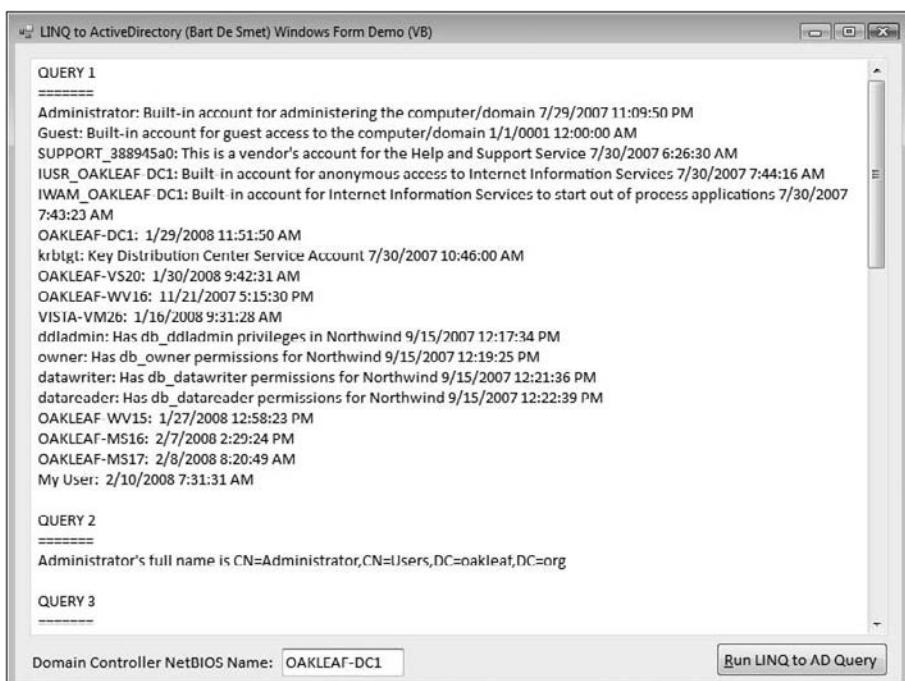


Figure 8-9

## Part III: Applying Domain-Specific LINQ Implementations

---

Client code must include handwritten classes for `DirectorySources` entities, such as `User` and `Group` in `Entities.cs` or `Entities.vb`, to define the type for `DirectorySource<T>` objects, which act as the data source for LINQ to LDAP queries.

*If you want to test LINQ to AD's ability to update AD property values with QUERY 7 and QUERY 8, add an Organizational Unit (OU) named Demo to the test domain and a User named MyUser with an Office location of Test. Code in the Entities.cs and Entities.vb files define the MyUser type, which has more properties than User defined.*

*BdsSoft.DirectoryServices.Linq and the sample query code have not been tested in production environments and should not be used to query or update production Active Directory domains.*

### LINQ to SharePoint

LINQ to SharePoint is another LINQ implementation by Bart De Smet. According to its CodePlex home page for the latest release when this book was written, 0.2.4.0 alpha of November 29, 2007 updated February 7, 2008, LINQ to SharePoint offers the following features:

- ❑ Custom query provider that translates LINQ queries to CAML
- ❑ Support for LINQ in C# 3.0 and Visual Basic 9.0
- ❑ Entity creation tool SpMetal to export SharePoint list definitions to entity classes used for querying
- ❑ Visual Studio 2008 integration for entity creation with SharePoint Model Language (SPML)
- ❑ Capability to connect to a SharePoint site either using the SharePoint object model or via the SharePoint Web services
- ❑ Planned support for updating through entity types

The programming model for LINQ to SharePoint is quite similar to that of LINQ to SQL. Both autogenerate XML mapping files and have designer files that contain autogenerated classes for source lists or tables. The primary difference is that LINQ to SharePoint's mapping file doesn't support a graphical editing mode.

*You can learn more about the LINQ to SharePoint and its 0.2.4.0 alpha release for VS 2008 from Bart De Smet's blog at <http://community.bartdesmet.net/blogs/linqtosharepoint>.*

*For more details about the SharePoint Lists Web service and CAML, read "Manage SharePoint Lists" from the February 2004 issue of Visual Studio Magazine (<http://visualstudiomagazine.com/columns/article.aspx?editorialsid=2104>), which includes a link to download the sample project's .NET 1.1 code.*

### Installing LINQ to SharePoint Assemblies

Downloading and running the current release, LINQ to SharePoint v0.2.4.msi when this book was written, from [www.codeplex.com/LINQtoSharePoint/](http://www.codeplex.com/LINQtoSharePoint/) installs the runtime code in \Program Files\BdsSoft LINQ to SharePoint and adds a LINQ to SharePoint File template to the Add New Item dialog's list.

## Chapter 8: Exploring Third-Party and Emerging LINQ Implementations

---

Version 0.2.4 used with VS 2008 requires running (as Administrator under Vista) the following commands from a Visual Studio 2008 Command Prompt to install four assemblies in the GAC:

```
cd %programfiles%\BdsSoft.Linq to SharePoint
gacutil -i BdsSoft.SharePoint.Linq.dll
gacutil -i BdsSoft.SharePoint.Linq.ObjectModelProvider.dll
gacutil -i BdsSoft.SharePoint.Linq.Tools.Spml.dll
gacutil -i BdsSoft.SharePoint.Linq.Tools.EntityGenerator.dll
```

Alternatively, execute the `InstallAssembliesInGAC.cmd` script from `\WROX\ADONET\Chapter08\LINQtoSharePoint`.

*If you download the source instead of runtime code, you'll need the Visual Studio 2008 SDK to compile the `BdsSoft.SharePoint.Linq` libraries.*

### Installing the SharePoint 3.0 SDK to Support SMPL

By default, LINQ to SharePoint uses SharePoint Web services to query lists. If you want to use the SharePoint Object Model and SharePoint Mapping Language (SMPL) instead of SharePoint Web services to query the lists, download and install the latest version of the SharePoint 3.0 SDK. (Search for "SharePoint 3.0 SDK" without the quotes.)

*To use the SharePoint Object Model, you must run the client locally on the server running the SharePoint site. This chapter's sample project uses SharePoint Web services.*

### Creating a Sample LINQ to SharePoint Project

If you don't have suitable lists to query in a test SharePoint site, use Microsoft Access 2007's Export to SharePoint feature to create them. Exporting the Products table to SharePoint 3.0 generates Products, Suppliers and Categories lists. If you have Access 2007 on your development machine, you'll be able to view and edit the list in a DataSheet ActiveX control.

*BdsSoft.SharePoint.Linq, related SharePoint libraries, and the sample query code have not been tested in production environments and **should not be used** to query or update production SharePoint sites.*

Start a new C# or VB project and add a LINQ to SharePoint File item to open the Add New Item dialog and give your `.spml` (XML) file a name, such as `NwindProducts`. Click Add to start the LINQ to SharePoint Entity Wizard that autogenerated a `.spml` file from lists on the site to serve as the SharePoint `DataContext` object. Click Next, type the SharePoint site's URL, and click the Test Connection button. Click Next to open the Define List Entities page and mark the check box for the lists you want to query (Categories, Products, and Suppliers for this example), and clear the check box for SharePoint-specific properties: `_ID`, `ContentType`, `Modified`, `Created` and `Version` (see Figure 8-10).

## Part III: Applying Domain-Specific LINQ Implementations

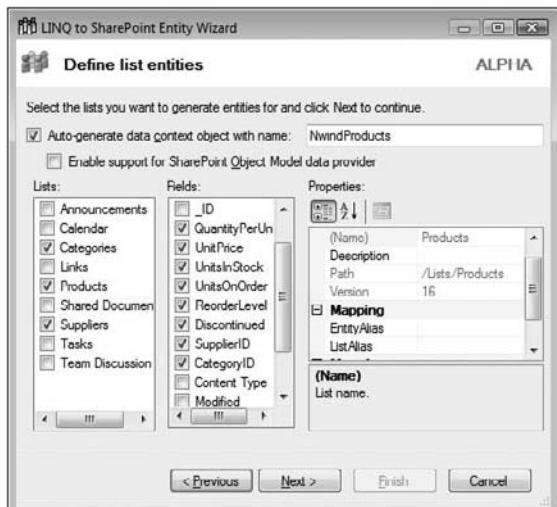


Figure 8-10

Click Next and Finish to create the mapping file and open it in VS 2008's XML Editor. Press F5 to compile and run the project, and then examine the `DataContextName.designer.cs` or `.vb` file that contains the autogenerated classes for the lists.

*A manual alternative to using the SharePoint Entity Wizard to create the C# or VB file is to use the SpMetal.exe command-line utility.*

### Writing LINQ to SharePoint Query Code

As noted in the “LINQ to SharePoint” introductory section, the code to instantiate `DataContext` objects as data sources and execute LINQ queries is almost identical to that for LINQ to SQL. Following is sample code from the `LINQtoSharePointC#.sln` and `LINQtoSharePointVB.sln` projects in the `\WROX\ADONET\Chapter08\C#\LINQtoSharepointC#` and `...\VB\LINQtoSharePointVB` folders:

#### C# 3.0

```
// Form-scoped DataContext
NwindProductsSharePointDataContext ctxNwind = null;

// Use constructor Uri overload with SharePoint Site URL from text box
Uri spUri = new Uri("http://" + txtServerName.Text);
ctxNwind = new NwindProductsSharePointDataContext(spUri);

var prods = from p in ctxNwind.Products
            where p.Discontinued == false
            orderby p.ProductName
            select new { ProductID = p.ProductID, ProductName = p.ProductName,
                        SKU = p.QuantityPerUnit, UnitPrice = p.UnitPrice,
                        Category = p.CategoryID.CategoryName,
                        Supplier = p.SupplierID.CompanyName,
                        Inventory = p.UnitsInStock, OnOrder = p.UnitsOnOrder };

// Populate a DataGridView with a List<AnonType>
```

## Chapter 8: Exploring Third-Party and Emerging LINQ Implementations

```
var lstProds = prods.ToList();
dgvLists.DataSource = lstProds;
```

### VB 9.0

```
' Form-scoped DataContext
Private ctxNwind As NwindProductsSharePointDataContext = Nothing

' Use constructor Uri overload with SharePoint Site URL from text box
Dim spUri As New Uri("http://" & txtServerName.Text)
ctxNwind = New NwindProductsSharePointDataContext(spUri)

' Include Supplier.CompanyName and Category.CategoryName from lookup lists
Dim Prods = From p In ctxNwind.Products -
    Where p.Discontinued = False -
    Order By p.ProductName -
    Select p.ProductID, ProductName = p.ProductName, _
        SKU = p.QuantityPerUnit, UnitPrice = p.UnitPrice, _
        Category = p.CategoryID.CategoryName, _
        Supplier = p.SupplierID.CompanyName, _
        Inventory = p.UnitsInStock, OnOrder = p.UnitsOnOrder

Dim lstProds = Prods.ToList()
dgvLists.DataSource = lstProds
```

The preceding query populates the sample projects' DataGridView controls, as shown in Figure 8-11, when you mark the Look Up Values check box and click the Product button. This query initially takes about 17 seconds to execute because of the lookup values; subsequent executions of the cached query take only about 300 ms on a moderately fast machine querying a remote SharePoint site on a 100-Mbps switched network.

	ProductID	ProductName	SKU	UnitPrice	Category	Supplier
▶	3	Aniseed Syrup	12 - 550 ml bottles	\$10.00	Condiments	Exotic Liqui
	40	Boston Crab Meat	24 - 4 oz tins	\$18.40	Seafood	New Engl
	60	Camembert Pierrot	15 - 300 g rounds	\$34.00	Dairy Products	Gai pâtu
	18	Carnarvon Tigers	16 kg pkg.	\$62.50	Seafood	Pavlova, L
	1	Chai	10 boxes x 20 bags	\$18.00	Beverages	Exotic Liqu
	2	Chang	24 - 12 oz bottles	\$19.00	Beverages	Exotic Liqu
	39	Chartreuse verte	750 cc per bottle	\$18.00	Beverages	Aux joyeu
	4	Chef Anton's Cajun Seasoning	48 - 6 oz jars	\$22.00	Condiments	New Orle
	48	Chocolade	10 pkgs.	\$12.75	Confections	Zaanse Sn
	92	Chu Hou Sauce	12 - 240 g jars	\$12.25	Condiments	Zhonghai
	38	Côte de Blaye	12 - 75 cl bottles	\$263.50	Beverages	Aux joyeu
	83	Cucumber Kimchi (Bag)	10 - 5 kg bags	\$225.00	Condiments	Seoul Kim
	82	Cucumber Kimchi (Jar)	12 - 1 kg bottles	\$42.50	Condiments	Seoul Kim
	58	Escargots de Bourgogne	24 pieces	\$13.25	Seafood	Escargots
	52	Filo Mix	16 - 2 kg boxes	\$7.00	Grains/Cereals	G'day, Ma
	71	Flotemysost	10 - 500 g pkgs.	\$21.50	Dairy Products	Norske M

Figure 8-11

## Part III: Applying Domain-Specific LINQ Implementations

---

Notice that CategoryName and SupplierName replace what you would expect to be CategoryID and SupplierID fields exposed by the Products list. Like the Entity Framework's Entity Data Model, foreign-key values are visible from LINQ lookup lists, Categories and Suppliers for this example.

### Translating Query Expressions to Method Calls and CAML

LINQ to SharePoint translates the C# query expression to the following method call chain:

#### C# 3.0

```
value(BdsSoft.SharePoint.Linq.SharePointList`1[LINQtoSharePointCS.Products]
    .Where(p => (p.Discontinued = Convert(False)))
    .OrderBy(p => p.ProductName)
    .Select(p => new <>f__AnonymousType0`8
        (ProductID = p.ProductID,
        ProductName = p.ProductName,
        SKU = p.QuantityPerUnit,
        UnitPrice = p.UnitPrice,
        Category = p.CategoryID.CategoryName,
        Supplier = p.SupplierID.CompanyName,
        Inventory = p.UnitsInStock,
        OnOrder = p.UnitsOnOrder))
```

And then uses an expression tree to translate the preceding SQOs into the following CAML expression, which the Web service client sends to the SharePoint instance in the request message:

#### CAML

```
<Query>
  <Where>
    <Eq>
      <Value Type="Boolean">0</Value>
      <FieldRef Name="Discontinued" />
    </Eq>
  </Where>
  <OrderBy>
    <FieldRef Name="Title" />
  </OrderBy>
</Query>
<ViewFields>
  <FieldRef Name="ID" />
  <FieldRef Name="Title" />
  <FieldRef Name="QuantityPerUnit" />
  <FieldRef Name="UnitPrice" />
  <FieldRef Name="CategoryID" />
  <FieldRef Name="SupplierID" />
  <FieldRef Name="UnitsInStock" />
  <FieldRef Name="UnitsOnOrder" />
</ViewFields>
```

The `<ViewFields>` group preserves column order specified by the `Select` projection.

The LINQ to SharePoint Query Visualizer includes two text boxes that display the preceding method call chain and the CAML request sent to the Lists Web service (see Figure 8-12).



Figure 8-12

To open the Query Visualizer, place a breakpoint on the `var lstProds = Prods.ToList()` or `Dim lstProds = Prods.ToList()` line, click one of the three buttons, and click the Data Tip's magnifying class.

## Summary

This chapter demonstrates that LINQ can be used to query a wide variety of data sources, not just in-memory objects, relational databases, XML documents, and DataSets. When this book was written, there were at least the following third-party LINQ implementations, in addition to those covered in this chapter: LINQ to LINQ to Amazon, LINQ to DataReader, LINQ to Expressions, LINQ to Google, LINQ to JavaScript, LINQ to LLBLGen Pro, LINQ to Lucene, LINQ to Mock, LINQ to MPI.NET, LINQ to Named Pipes, LINQ to OneNote, LINQ to Outlook, LINQ to Reflection, LINQ to Streams, LINQ to TerraServer, and LINQ to WMI. (LINQ to TerraServer is a demo application by Microsoft employees Charlie Calvert, Alex Turner, and Mary Deyo.) The OakLeaf Systems blog (<http://oakleafblog.blogspot.com>) has links to more information about — and often the source code for — the preceding implementations in the left column's "Labels" section. New entries appear almost every month.

*Oren Novotny, this book's technical editor, developed LINQ to Streams (also known as Streaming LINQ or SLinq) to process data on the fly, such as that supplied by financial data syndicators. You can learn more about LINQ to Streams at its CodePlex site, [www.codeplex.com/Sling](http://www.codeplex.com/Sling).*

As you would expect, Microsoft has the most domain-specific LINQ implementations in active development, including Parallel LINQ and LINQ to REST, which this chapter covered in detail. LINQ to XSD and LINQ to Stored XML projects were in the incubator stage when this book was written, and both are expected to surface as Web releases of Beta or CTP versions in 2008. ADO.NET Data Services moved from incubator to formal project status with substantial staffing in mid-2007 and became one of the "pillars of the Entity Data Platform" by the end of the year. Volta, another incubator project once referred

## Part III: Applying Domain-Specific LINQ Implementations

---

to by its creator, Erik Meijer, as “LINQ 2.0”, emerged as a CTP in early December 2007. However, Volta has no direct dependency on LINQ.

Bart De Smet was one of LINQ adopters and created a third-party, open-source implementation of the LINQ to Objects SQOs to compensate for the disappearance of Microsoft’s source code for the SQOs when LINQ joined the VS 2008 beta program. You can download the source code for the .NET 3.5 and VS 2008 RTM bits version of the BdsSoft.Linq.dll library from CodePlex at [www.codeplex.com/LINQSQO/Release/ProjectReleases.aspx?ReleaseId=8568](http://www.codeplex.com/LINQSQO/Release/ProjectReleases.aspx?ReleaseId=8568). The project includes more than 500 unit tests.

De Smet’s LINQ to Active Directory and LINQ to SharePoint are open-source “labor of LINQ-love” projects that illustrate the programming skills necessary to take advantage of `IQueryable<T>` query representations. `IQueryable<T>` enable writing expression trees to translate LINQ SQOs to other query languages, such as LDAP filters and CAML messages for SharePoint’s List Web services. The chapter concluded with examples that use the `BdsSoft.DirectoryServices.Linq` and `BdsSoft.SharePoint.Linq` with Active Directory domain controllers and SharePoint sites.

# **Part IV: Introducing the ADO.NET Entity Framework**

**Chapter 9:** Raising the Level of Data Abstraction with the  
Entity Data Model

**Chapter 10:** Defining Storage, Conceptual, and Mapping Layers

**Chapter 11:** Introducing Entity SQL

## Part IV: Introducing the ADO.NET Entity Framework

The Entity Framework (EF) is an object persistence layer that includes an object/relational mapping (O/RM) tool to substitutes an object graph diagram for the traditional relational database schema. The duties of a persistence layer are to instantiate in memory a desired set of object instances together with a representation of their associations and save in-memory changes to objects and their property value to rows and columns of relational tables. What distinguishes EF from the 40-plus commercial and open-source O/RM tools for .NET is an intermediate mapping from the relational data model to the entity-relationship (E-R) data model proposed by Dr. Peter Chen in 1976 and called the Entity Data Model (EDM) by EF. Three XML files, a storage schema (*ModelName.ssdl*), mapping layer (*ModelName.msdl*), and conceptual schema (*ModelName.csdl*) together with EF runtime code perform the object-to-relational mapping. The storage schema has a 1:1 relationship with the tables, keys, and relationships of the underlying database, called the *persistence store*. The conceptual schema represents the transformation by the mapping layer of the storage schema to a set of *EntityType*s, *EntitySet* collections, and *AssociationSet* collections with Common Language Runtime (CLR) data types. Thus the conceptual schema isn't locked into a 1:1 relationship with the storage.

To ease the process of defining the tasks performed by the mapping layer, EF includes a graphical Entity Data Model Designer to display and edit *EntityType*s, properties, and *Associations*, which the EDM calls *Navigation Properties*. The EDM Designer automatically generates partial classes in a C# or VB file to implement these objects for a Windows or Web form project. The data source for the EDM Designer is a single XML file, *ModelName.edmx*, which combines the contents of the three mapping files.

Figure IV-1 shows the designer's surface for the Northwind database's Orders and Order Details tables and the Association between them. The Model Browser window at the right of Figure IV-1 displays nodes for the two *EntityType*s with property items expanded for *Order\_Detail*, *EntitySets*, *AssociationSets*, and the tables and constraints of the data store. The Mapping Details window at the bottom of the figure maps *Order Details* table fields to *Order\_Detail* *EntityType* properties.

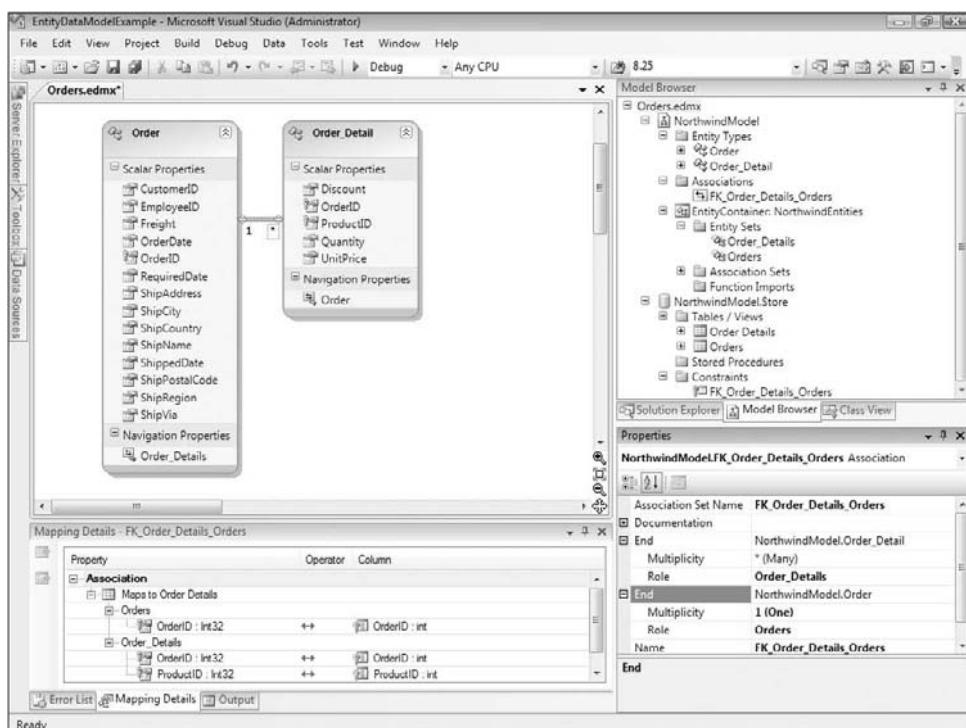


Figure IV-1

*LINQ to SQL, which is the subject of Chapter 5, “Using LINQ to SQL and the LinqDataSource,” automatically singularizes the names of EntityTypes when it encounters tables with plural names, but EF doesn’t. EntityType names in chapters of parts IV and V are manually singularized and EntitySet names substitute a plural suffix for the . . . Set suffix that the designer adds automatically after singularization of EntityType names.*

The theoretical benefit of a mapping layer with human-readable files is that changes to the database’s schema can be isolated from the EDM, although effective schema-EDM isolation is seldom achieved in the real world. Adding or removing a table, field, relationship, or foreign-key constraint usually requires the developer to rewrite at least a part of the project’s source code. The EDM Designer includes an Update from Database feature that can recreate the schema and mapping files, but doing so removes all custom features from the XML mapping files.

## A Brief History of Entity Framework’s Development

The first glimmer of what was to become EF appeared in the “DAT200: Future Directions of Data Driven Design” session, presented by Microsoft Technical Fellow David Campbell at Microsoft’s Professional Developers Conference (PDC) in October 2005. Pablo Castro, who’s now an architect for ADO.NET Data Services, gave a demonstration of early versions of the EDM Designer and XML mapping files. The session’s name is ironic because the fact that Entity Framework has a *data-driven* (also called *data-first*) design instead of a *domain-driven* design is a very controversial issue.

Microsoft filed a patent application on the EDM’s basic elements as an “Incremental Approach to an Object-Relational Solution” on February 28, 2006; the U.S. Patent and Trademark Office published it as number 20070055692 on March 8, 2007. The ADO.NET Team, which is part of the SQL Server Data Programmability (DP) group, released the first EF Community Technical Preview (CTP) in mid-August 2006; it was followed by an Orcas October 2006 CTP and June 2007 CTP without the EDM Designer. The team originally expected to ship EF with VS 2008 when the latter released to manufacturing (RTM) in November 2007. However it became evident by the end of April 2007 that EF and the EDM Designer wouldn’t be ready in time, and DP Architect Mike Pizzo announced in an April 28, 2007 ADO.NET Team blog post:

Based on the need to align with requirements from key internal partners that are building on the Entity Framework, along with the need for a better tool experience, we have decided to ship the ADO.NET Entity Framework and Tools during the first half of 2008 as an update to the Orcas release of the .NET Framework and Visual Studio.

In the meantime, a group of Microsoft Most Valuable Professionals (MVPs) gathered in Seattle and Redmond in mid-March 2007 for the 2007 MVP Global Summit. Several MVPs, who were well-known .NET developers and later became known as the “NHibernate Mafia,” took members of the ADO.NET team to task for alleged deficiencies in EF’s design:

- ❑ EF maps from the database schema to the domain model, which violates the principles of domain-driven design.
- ❑ Classes that represent entities must inherit from the mapping base class, which prevents the domain model from being “persistence ignorant” and exposes domain entities to other tiers.
- ❑ Change tracking requires each entity property’s `Set` method to raise a “`PropertyChanged`” event.

## Part IV: Introducing the ADO.NET Entity Framework

---

The “Understanding the Persistence Ignorance Controversy” topic of Chapter 9, “Raising the Level of Data Abstraction with the Entity Data Model,” offers an analysis of persistence ignorance as it applies to EF and other O/RM tools.

The team then released EF Beta 2 and EDM Designer CTP 1 at the end of August 2007; EF Beta 3 and EDM Designer CTP 2 followed in early December 2007. The VS 2008 SP1 beta, which became available on May 12, 2008, included updates to EF and the EDM Designer but did not settle the issues raised by the MVPs. VS 2008 SP1 released to the Web on August 11, 2008 and, therefore, EF missed Pizzo’s “first half of 2008” target ship date by about 2.3 months.

## Entity Framework’s Future

Tim Mallilieu, an EF Program Manager, posted on June 23, 2008 an initial “Transparency in the Design Process” entry to the new Entity Framework Design blog (<http://blogs.msdn.com/efdesign/archive/2008/06/23/transparency-in-the-design-process.aspx>). This post outlined the following new features the team intended to include in EF v2:

- ❑ Persistence ignorance with Plain Old CLR Objects (POCO)
- ❑ N-Tier support using standard Windows Communication Foundation (WCF Services)
- ❑ Domain-first as an alternative to data-first design, with the ability to generate and deploy a database from the domain design
- ❑ Test-driven development (TDD) support with POCO classes and virtual ObjectQuery<T> context members by eliminating the requirement to use the database as a data source when executing unit tests
- ❑ Foreign-key properties to extend Navigation Properties in the model and POCO classes
- ❑ Implicit lazy loading as an alternative to explicit lazy loading by executing the Load() method
- ❑ Query tree rewriting to enable filtering query results vertically and horizontally

In addition, Mallilieu adopted the “transparent design process” pioneered for ADO.NET Data Services (a.k.a., “Astoria”) by architect Pablo Castro, which involves posting minutes of design meetings in an MSDN blog, actively seeking developer design recommendations, and releasing frequent CTPs during EF v2’s evolution. The blog for EF v2 is Entity Framework Design (<http://blogs.msdn.com/efdesign>).

Microsoft also established a Data Programmability Advisory Council consisting of the following luminaries of software patterns and domain-driven design:

- ❑ **Eric Evans**, author of *Domain-Driven Design: Tackling Complexity in the Heart of Software* (<http://www.domainlanguage.com/about/ericevans.html>)
- ❑ **Stephen Forte**, chief strategy officer at Telerik and Microsoft regional director (<http://www.stephenforte.net>)
- ❑ **Martin Fowler**, author of *Patterns of Enterprise Application Software* and many other books, and originator of the term *Persistence Ignorance* (<http://martinfowler.com>)

- Pavel Hruby**, author of *Model-Driven Design Using Business Patterns* and the Visio Stencil for UML and SysML (<http://www.phruby.com>)
- Jimmy Nilsson**, author of *Applying Domain-Driven Design and Patterns with Examples in C# and .NET* and maker of the rules for Persistence Ignorance (<http://jimmynilsson.com>)

The first meeting occurred in July, 2008 but the minutes of this meeting, which Mallalieu promised to publish, had not appeared by the time this book was written.

# The ADO.NET Entity Framework

## Vote of No Confidence

The same day that Mallilieu posted his “Transparency in the Design Process” blog entry, an anonymously authored “ADO.NET Entity Framework Vote of No Confidence” manifesto appeared on the Web as an electronic form (<http://efvote.wufoo.com/forms/ado-net-entity-framework-vote-of-no-confidence>) to which users could append their names. The petition focuses on these five theses:

- EF inordinately focuses on the data aspect of entities leads to degraded entity architectures
- Lack of persistence ignorance causes business logic to be harder to read, write and modify, causing development and maintenance costs to increase at an exaggerated rate
- Excess code is needed to deal with lack of lazy loading
- A shared, canonical model contradicts software best practices
- Excessive merge conflicts occur with source control in team environments

Subsequent events disclosed that the authors of the manifesto were participants in the confrontation at the 2007 MVP Global summit, but only the first two of the preceding topics duplicate issues. Mallalieu addressed the first three issues with the new features planned for EV v2 in his “Vote of No Confidence” post in his personal MSDN blog (<http://blogs.msdn.com/timmall/archive/2008/06/24/vote-of-no-confidence.aspx>). “Shared, canonical model” isn’t commonly used in conjunction with domain modeling, but Mallalieu offered a comment about the advantage of a common EDM for relational data, reporting services, extract/transform/load (ETL) operations, and data synchronization.

A “Vote of No Confidence” won’t affect the DP group’s determination to make EF the .NET data platform of the future and replace `DataSets` with `EntitySets`. The handwriting is on the wall, as you see in the chapters of this part and the next.



# 9

## Raising the Level of Data Abstraction with the Entity Data Model

Microsoft's ADO.NET Entity Framework (EF) represents a seismic shift in the SQL Server Data Programmability group's approach to querying and updating relational data. Since the initial release of Visual Studio .NET in 2002, the `DataSet` has been central to Microsoft's data management toolset strategy. The `DataSet` represents an in-memory cache of relational `DataTable`(s) that programs can populate from and persist to XML documents in a file system or tables in a relational database management system (RDBMS).

`DataSets` extended the “disconnected client” architecture introduced by COM-based ADO `Recordsets` to an extensive ecosystem of graphic designers for generating C# or VB classes to define strongly typed `DataSets`,  `DataTables` and `DataViews`, databinding components and data-bound controls. Wizards orchestrated combinations of these elements to autogenerated simple Windows form projects that enabled displaying and editing relational data in bound text boxes and data grids. The ability to create a working — but trivial — database application with 20,000 or more lines of code in a minute or two was a big hit among most attendees at Microsoft conferences but didn't impress the more jaded developers of data-intensive projects at the enterprise level.

While the Data Programmability group was investing heavily in developer tools for relational data, the Developer Division's C#, VB, and VS teams were emphasizing object-oriented (OO) programming techniques and tools that would enable Microsoft programming languages and Visual Studio to compete more effectively with Java. Hibernate 1.0, which has become (along with Sun Microsystem's Java Persistence API) a leading object-relational (O/R) persistence and query

## Part IV: Introducing the ADO.NET Entity Framework

---

tool for Java, released as an open-source project from SourceForge.Net on July 5, 2002 with the following description:

Hibernate is a powerful, high performance object/relational persistence and query services [framework] for Java. Hibernate lets you develop persistent objects following common Java idiom, including association, inheritance, polymorphism, composition and the Java collections framework. No code generation or bytecode processing is required. Instead, in pursuit of a shorter build procedure, runtime reflection is used. Hibernate supports Oracle, DB2, MySQL, PostgreSQL, Sybase, Interbase, Microsoft SQL Server, Mckoi SQL, Progress, SAP DB and HypersonicSQL.

*The “LINQ to SQL” section of the Introduction to Part III and the first part of Chapter 5 describe OO development platforms’ need for easy-to-use, prebuilt, predictable object/relational persistence and query services.*

Microsoft’s Professional Developers Conference (PDC) 2001 delivered a technical preview of ObjectSpaces, which was intended to offer .NET developers object/relational persistence and query services similar to Hibernate. Like Hibernate, which has its own query dialect, HQL, ObjectSpaces provided OPath, a query language modeled on XPath. As mentioned earlier in the book, ObjectSpaces was to be a component of VS 2005 but was subsumed into WinFS and then discarded when WinFS was cut from Vista and Longhorn Server before the latter was renamed Windows Server 2008.

*JBoss, now a subsidiary of Red Hat Inc., released NHibernate 1.0, a port of Hibernate 2.1 to .NET 1.1 and 2.0 to SourceForge on October 19, 2005. Many .NET developers adopted NHibernate as their O/RM tool of choice, and several .NET Most Valued Professionals (MVPs) are vocal proponents of NHibernate. The “Understanding the Persistence Ignorance Controversy” section near the end of this chapter describes the primary issues the MVPs have raised with EF v1. Although Hibernate and NHibernate are open-source projects, Red Hat has registered both trademarks and employs developers to work on both products, for which the company offers paid support services.*

EF is the heir apparent to DataSets as Microsoft’s preferred relational data management architecture. EF is much more than a conventional object/relational mapping (O/RM) tool. EF maps the schema of an underlying relational database to an *Entity Data Model* (EDM), which is based on Dr. Peter Chen’s *Entity-Relationship* (E-R) model. Like ObjectSpaces, three XML files define the physical (relational), mapping and conceptual (E-R) schemas of layers to make the transformation from the relational model to the EDM. An SQL-like language called *Entity SQL* (also called eSQL) queries the EDM’s *mapping provider* and returns to the `DbDataReader` a collection of `DbDataRecords` by default. An Object Services layer enables Entity SQL queries against `ObjectQuery` instances to return `IQueryable` collections of entities (`EntitySets`) or anonymous types instead of `DbDataRecords`. Substituting LINQ to Entities for Entity SQL queries against `ObjectQuery` instances is an alternative method for returning `EntitySets` or anonymous types.

*Dinesh Kulkarni, a program manager for ObjectSpaces and LINQ to SQL, stated in a September 13, 2005 blog post, “The future of ObjectSpaces is DLinq [the early codename for LINQ to SQL]. We used the feedback we got on ObjectSpaces to design DLinq as a better way to query database to get objects and to persist them back to the database.” In reality, the “future of ObjectSpaces” is Data Programmability’s EF. Kulkarni and two other ObjectSpaces team members joined the C# team after ObjectSpaces disappeared to create LINQ to SQL as a domain-specific LINQ demonstration. (The C# team “owns” LINQ.) EF’s three-tier mapping file structure, multiple query languages, and support for a variety of RDBMSs contrast with LINQ to SQL’s attribute-based mapping with classes, plain old XML objects (POCO) for entities with an optional XML mapping file, LINQ-only query language, and dedication to SQL Server 200x, SQL Server Express, and SQL Server Compact Edition databases only.*

## Chapter 9: Raising the Level of Data Abstraction

---

The Data Programmability group is planning a grandiose data access strategy called the *Entity Data Platform* that the group's architect, Michael Pizzo, described in an April 28, 2007 blog post (<http://blogs.msdn.com/data/archive/2007/04/28/microsoft-s-data-access-strategy.aspx>) as follows:

Microsoft envisions an Entity Data Platform that enables customers to define a common Entity Data Model across data services and applications. The Entity Data Platform is a multi-release vision, with future versions of reporting tools, replication, data definition, security, etc. all being built around a common Entity Data Model.

Within the .NET Framework, the ADO.NET Entity Framework is integral to this vision. The ADO.NET Entity Framework builds on our mutual investment in ADO.NET by enabling applications to write to a conceptual data model with strong notions of type, inheritance, and relationships.

EF is the preferred data source for .NET Data Services, ASP.NET Dynamic Data, and other new data-intensive Microsoft technologies, and currently is the only data source for `IEnumerable<T>` sequences that has the potential for use with RDBMSs other than SQL Server 200x or SQL Server Express and Compact editions. It is the only Microsoft O/RM tool that has the capability to handle business object polymorphism with something other than the table-per-hierarchy (TPH) inheritance model. The `EntityDataSource` is a new ASP.NET server control that emulates for LINQ to Entities only the `LinqDataSource` control for LINQ to SQL. Chapter 15, "Using the Entity Framework As a Data Source" covers new EF-related technologies and the `EntityDataSource` control.

This chapter provides an introduction to the objectives, architecture, and implementation of EF, EDM and their elements with relatively simple code examples. The remaining chapters of Part IV describe EF's components in greater detail.

## Understanding the Entity-Relationship Model

Dr. Peter Chen presented his "The entity-relationship model — toward a unified view of data," paper at the International Conference on Very Large Databases in September, 1975. The inaugural issue (Vol. 1, No. 1) of the *ACM Transactions on DataBase Systems* of March 1976 reprinted the paper, which you can read at <http://csc.lsu.edu/news/erd.pdf>.

*Dr. Chen's presentation followed by five years Dr. E. F. Codd's seminal "A relational model for large shared data banks" paper, which the Communications of the ACM published in June 1970 and that led to the development of today's RDBMSs.*

Dr. Chen claimed in the paper that his entity-relationship model (E-RM):

- ❑ “[A]adopts the more natural view that the real world consists of entities and relationships.”
- ❑ “[I]ncorporates some of the important semantic information about the real world.”
- ❑ “[C]an be used as a basis for a unified view of data [stored in network, relational, and entity set models].”

## Part IV: Introducing the ADO.NET Entity Framework

---

E-R and E-R diagrams have been widely adopted by data-oriented architects and developers, and form the basis of many database design tools, such as the CA ERwin Data Modeler. Microsoft Office Visio Professional and higher support E-R and the more recent Object Role Modeling (ORM) diagrams. Based on the widespread use of E-R diagrams for more than 25 years, it's safe to say that the preceding claims have proven true.

### **Entity-Relationship and EDM Terminology**

Correctly interpreting E-R and EDM terminology is critical to understanding how to implement EF-based projects. The EF development team adopted much of the EDM's terminology (shown below in monospace) from Dr. Chen's paper (in italics):

- ❑ *Entities* are “things” that can be distinctly identified, such as persons, organizations and events. Documents representing objects, such as deeds, and transactions, sales orders, purchase orders, invoices, and line items also are entities. EDM refers to entities as `EntityType`s, which the EF documentation describes as “abstract specifications for the details of a data structure in the application domain.” EDM uses table names for `EntityType` names, but the singular form is preferred; fortunately, editing `EntityType` names in the graphical EDM Designer is easy.
- ❑ *Entity sets* classify (contain) entities with common properties. EDM's `EntitySets` contain `EntityType`s of the same type or a subtype. It's a common practice to name EDM `EntitySets` with plural of the `EntityType`; Dr. Chen uses the singular form. An EDM `EntityContainer` represents a database, which is a collection of `EntitySets`.
- ❑ *Entity primary keys* uniquely identify each entity in an entity set. An entity primary key, which EDM calls a *Key*, *Key attribute*, or *Key value*, may be a natural (semantically meaningful) or surrogate (artificial) key, such as an auto-incrementing `int identity` or `ROWGUIDCOL` column.
- ❑ *Relationships* represent *associations* between entities, which EDM calls `Associations`. Most `Associations` are binary (between two different entities), but unary (between different members of the same entity, such as an `Employee` who reports to another `Employee` as his or her manager) and ternary (between three entities, such as `Supplier-Product-OrderItem`).
- ❑ *Relationship sets* are tuples of relationships. The mathematic definition of tuple is *an ordered list of objects of a specified type*. `IEnumerable<AnonymousType>` collections returned by LINQ queries are *n*-tuples, where *n* is the number of the projection's members. A quadruple results from a four-member projection. EDM calls the tuples `AssociationSets` and names them from the underlying foreign-key constraint name, typically `FK_Source_Target` for SQL Server where `Source` is the `EntityType` name of the `End` that represents the 1 side of a 1:many relation and `Target` is the many `End`.
- ❑ *Roles* are the function that the entity performs in the relationship, such as Husband and Wife for Person entities in a Marriage relationship. An EDM `Association` represents each associated `EntityType` as an `End` that has a `Role` attribute.
- ❑ *Mappings* specify the potential number of entities that can participate in the relationship, which more commonly is called cardinality. Cardinality can be *1:n* (one-to-many, such as `order:lineitems`), *m:1* (many-to-one, such as `customer:orders`) and *m:n* (many-to-many, such as `products:suppliers`.) EDM substitutes a `Multiplicity` attribute for each `End` of an `Association`: *1* represents that a single `EntityType` is required, *0...1* indicates that a single `EntityType` is optional, and *\** represents many.

- ❑ *Attributes* are functions that map from an entity set to a *value set*; for example an *OrderDate* attribute maps to a Date/Time value set, which (for SQL Server) ranges from 1753-01-01T00:00:00 to 9999-12-31:23:59:59. EDM calls an attribute a *Property*, instead of a field or column.

*Chen calls an entire table an entity relation and a row an entity tuple. Relationship sets are represented by a tabular data structure consisting of rows having attribute values of the entity primary key for each role called relationship tuples. EDM doesn't use tuple terminology.*

### Entity-Relationship Diagrams

EF's graphical EDM designer enables creating simplified entity-relationship diagrams, which are similar to those generated by LINQ to SQL's O/R Designer and described in Chapter 5. Classic E-R diagrams represent entity sets by rectangular-shaped boxes and relationship sets by diamond-shaped boxes, as shown in Figure 9-1 for part of the Northwind sample database. A crow's-foot symbol on the line connecting the entity box to the relationship diamond represents *many*; a dash crossing the line represents *1* and a circle on the line represents *0*. Therefore, symbols define cardinality as follows:

- ❑ A ring and dash represent zero or one
- ❑ Two dashes represent exactly one
- ❑ A ring and crow's foot represent zero or more
- ❑ A dash and crow's foot represent one or more

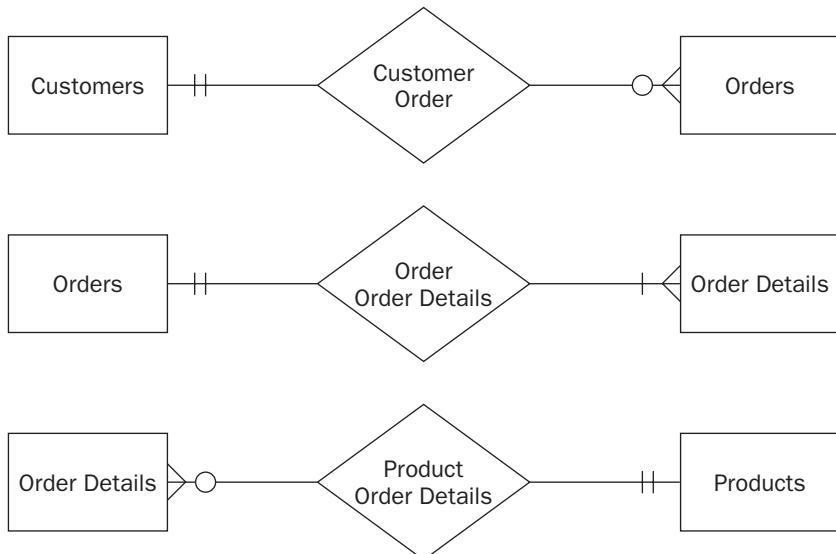


Figure 9-1

## Part IV: Introducing the ADO.NET Entity Framework

Modern E-R diagrams usually omit the diamond-shaped box and list every attribute in the entity set box, identifying entity-primary-key attributes with PK and bold font. When diagramming relational data by reverse-engineering in Microsoft Office Visio 2007, foreign-key fields have an FK prefix, as shown in Figure 9-2 and foreign-key relationships terminate with arrows. EF's EDM Designer doesn't include foreign-key fields in EntitySet Property lists, as you'll see shortly.

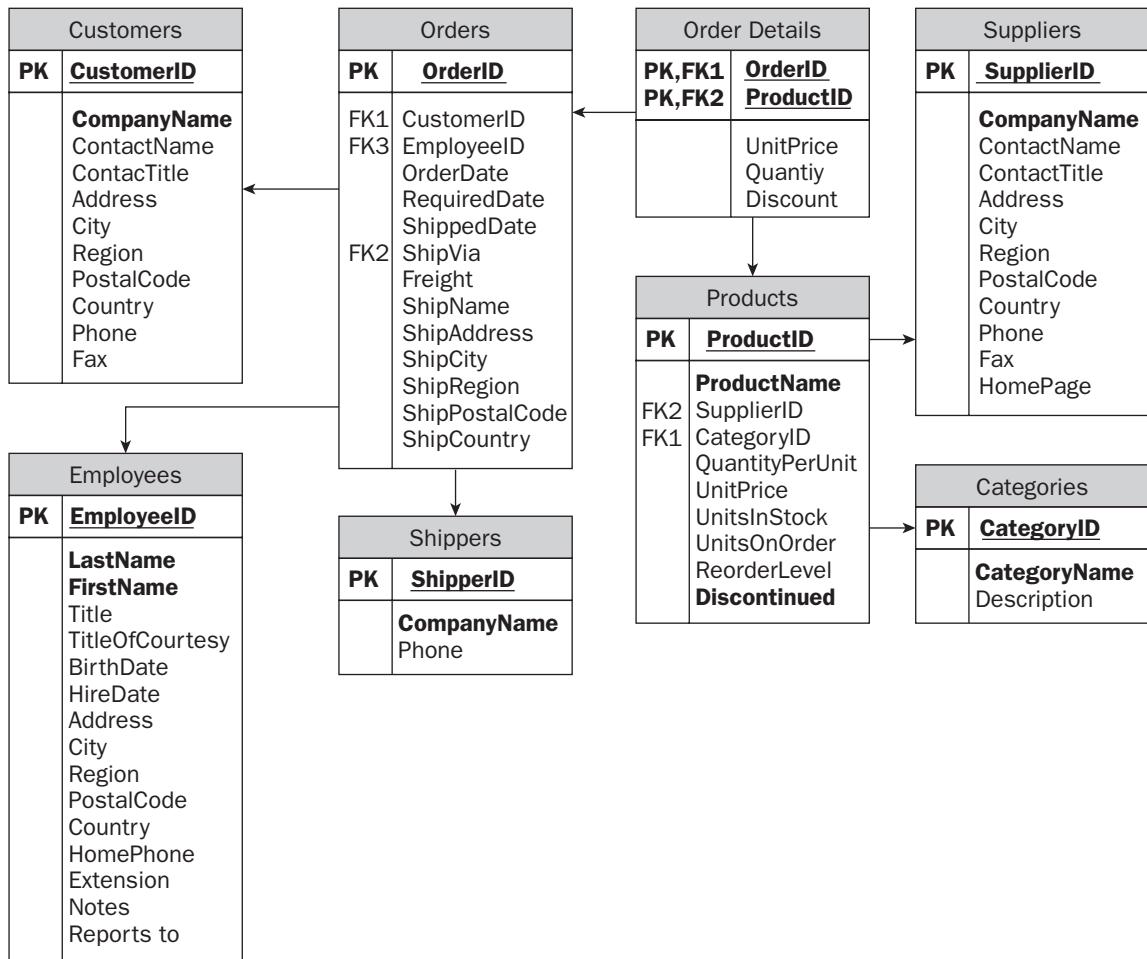


Figure 9-2

SQL Server's diagramming tool that's part of the SQL Server Management Studio [Express] toolset creates entity relationship diagrams based on foreign-key constraints automatically. These diagrams express 1:many relationships with 1 and  $\infty$ , respectively, and don't support cardinality symbols. You must specifically enable adding an E-R diagram to a SQL Server 2005+ database and remember to exclude its table when adding tables to the EDM Designer.

# Comprehending Entity Framework Architecture and Components

The Entity Framework implements and exploits the Entity Data Model with the following four major components:

- ❑ **Mapping files and EDM Designer.** Three XML files manage mapping for the transition from the physical (relational or store) schema to the conceptual (business-object-oriented) schema. Defining mapping in XML files minimizes — but doesn't eliminate in v1 — persistence-related attributes in business-object classes and prevents the need for reflection at runtime. The graphical EDM Designer autogenerated a single `ModelName.edmx` file and the top-level `ObjectContext` and `EntityName` business-object classes, whose use is optional, in `ModelName.Designer.cs` or `ModelName.Designer.vb`. Building the project creates the three mapping XML files, `ModelName.ssdl`, `ModelName.msl`, and `ModelName.csdl` and stores them in the assembly as resources.
- ❑ **EntityClient, Entity SQL, and Client Views.** `EntityClient` is an ADO.NET data provider that processes entity queries written in an entity-enabled SQL dialect called Entity SQL to generate Client Views with a `DbDataReader`. `EntityClient` emulates `SqlClient`'s properties with `EntityConnection`, `EntityCommand`, `EntityDataReader`, and other `Entity...` properties and objects. The syntax of Entity SQL queries is identical for all database brands that have a storage-specific ADO.NET data provider, such as the current `SqlClient` version. `EntityClient` hands off its Canonical Query Tree (CQT, an expression tree, sometimes called a Canonical Command Tree or CCT) to the storage-specific data provider, which translates the CQT to the RDBMS's SQL dialect.
- ❑ **Object Services.** Object Services is a layer over the `EntityClient` that provides an `ObjectQuery` instance to execute Entity SQL statements that return strongly typed business object instances instead of high-performance Client Views. The `ModelName.Designer.cs` or `ModelName.Designer.vb` file defines the business classes at design time. Use of the Object Services layer is optional.
- ❑ **LINQ to Entities Provider.** LINQ to Entities is a domain-specific LINQ implementation that uses an `IQueryable<T>` expression to generate an expression tree, which the `EntityClient` translates to a CQT and passes to the storage-specific data provider. The storage-specific provider returns a `DbDataReader`, which `EntityClient` converts to a `IEnumerable<T>` sequence for the data access or presentation layer. In this respect, LINQ to Entities parallels LINQ to SQL's query pipeline that uses expression trees to output T-SQL.

*The first two of the preceding components are common to the Entity Data Model in its role as a foundation for future members of the nascent Entity Data Platform.*

Figure 9-3 is a diagram of the relationship between the data access or presentation layer, EF's components, the storage-specific ADO.NET data provider, and the RDBMS.

## Part IV: Introducing the ADO.NET Entity Framework

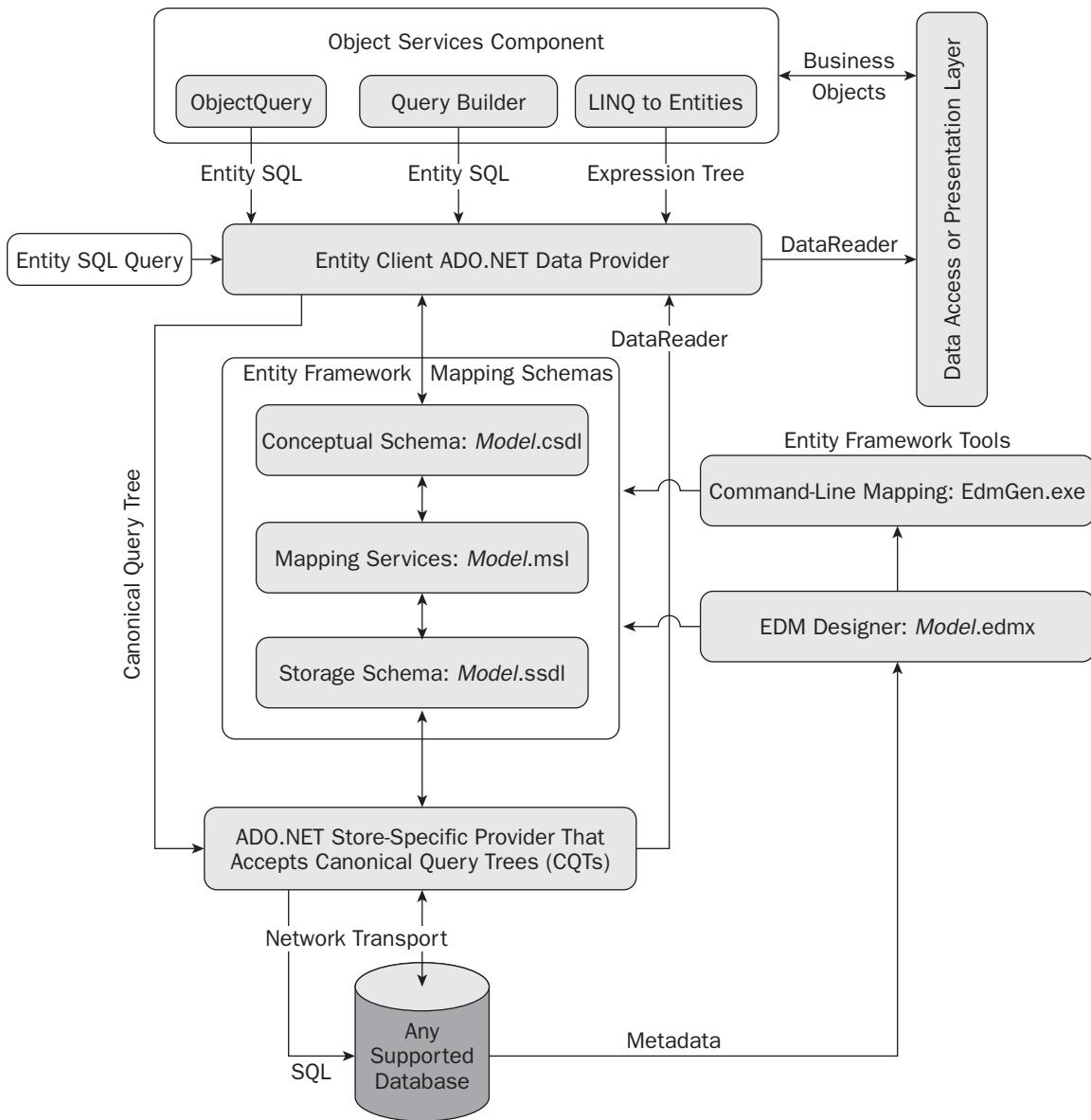


Figure 9-3

*Entity SQL v1 doesn't support SQL Data Manipulation Language (DML) keywords (INSERT, UPDATE, and DELETE) for updating the persistence store. Updating the store requires bringing the entity instance to update or delete into memory, modifying its properties or deleting it, and then saving the changes to the persistence store. Inserting a new instance requires instantiating a new object, setting its properties, and saving it to the persistence store.*

### **Mapping from the Physical to the Conceptual Layer with the EDM Designer**

EF provides a command-line application named `EdmGen.exe` for creating the three XML mapping files and the object layer class file. `EdmGen.exe` has an arcane command-line syntax and generates objects for every table in the target database. Fortunately, there's an Entity Data Model Wizard that lets you select a Data Connection from Server Explorer and then specify the tables, views, or stored procedures to add to your EDM.

*The sample projects for the following sections are `NwindOrdersEdmCS.sln` and `NwindOrdersEdmVB.sln` in the `\WROX\ADONET\Chap09\CS` and ... \VB, respectively. The `Northwind.cs` and `Northwind.vb` files contain the Object Layer classes.*

### **Creating the XML Mapping Files and Object Layer Class File with the EDM Wizard**

Installing EF v1 from the Web release adds an ADO.NET Entity Data Model item template to the Add New Item dialog. Add this item as a `ModelName.edmx` file, `Northwind.edmx` for this example, to a Windows or Web form project to start the Entity Data Model Wizard. Accept the default Generate from Database option in the Wizard's Choose Model Contents dialog, and click Next to enable the EDM as a data source from the RDBMS whose connection you select in the Choose Your Data Connection dialog (see Figure 9-4).

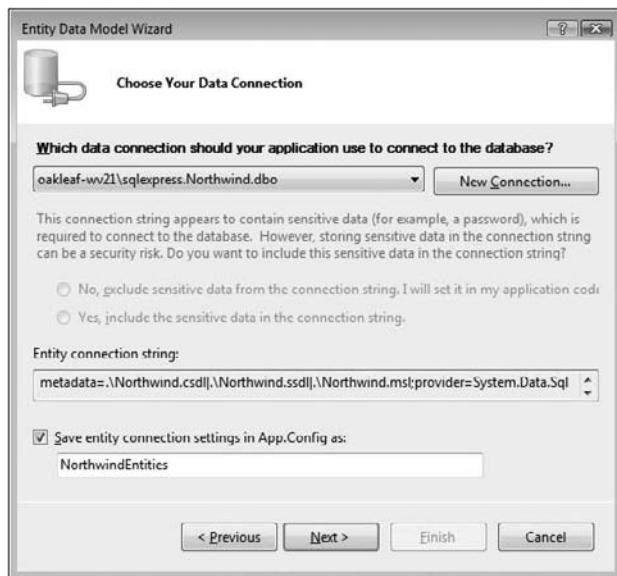


Figure 9-4

*You can design your entity model from scratch by selecting the Empty option and dragging Entity, Association, and Inheritance controls from the toolbox to an empty EDM Designer surface. Generating the persistence store from the business entities you design is the preferred approach for Domain-Driven Design (DDD). However, EF v1 won't generate a store database from the design, so this option wasn't practical when this book was written. The ADO.NET team has scheduled building the database from the EDM for v2.*

## Part IV: Introducing the ADO.NET Entity Framework

The EDM Wizard is similar to the LINQ to SQL Wizard except for its connection string and default connection name. Following is a typical connection string for a Windows form project with a local SQL Server 2005 Express instance:

```
metadata=res://*/Northwind.csdl|res://*/Northwind.ssdl|res://*/Northwind.msl;
provider=System.Data.SqlClient;provider connection string="Data
Source=localhost\SQLEXPRESS;Initial Catalog=Northwind;Integrated Security=True"
```

The `res://*/` prefix designates storing the mapping files as an assembly resource. The “Locating Mapping Files” section of Chapter 10 provides more information on connection strings for specific application types.

The `metadata` attribute defines the names and location for the three XML mapping files; `provider` designates the store-specific ADO.NET data provider (`SqlClient` for SQL Server) and `provider connection string` represents the connection string for the data provider. The connection string name you choose becomes the name of the top-level `EntityContainer` object, the default `NorthwindEntities` for this example.

Click `Next` to open the Choose Your Database Objects dialog in which you select the databases tables, views, and stored procedures to map to the conceptual layer. By default the wizard selects all objects of these types and assigns `DatabaseNameModel` as the EDM’s default namespace, `NorthwindModel` in this case; for brevity, this example uses only the Northwind Orders and Order Details tables (see Figure 9-5).

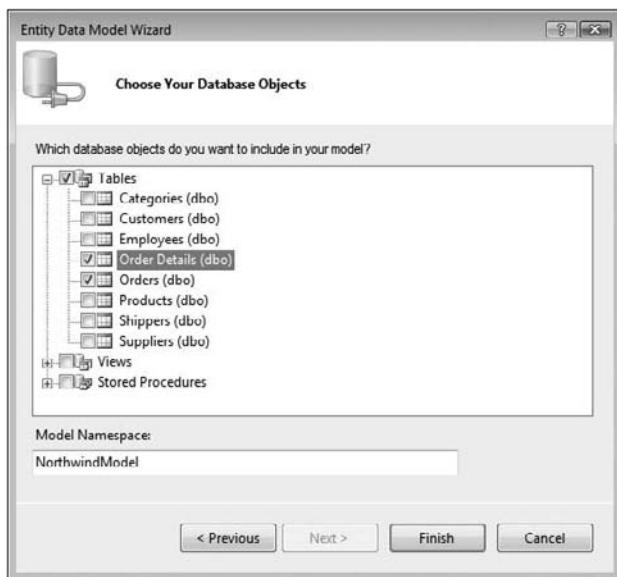


Figure 9-5

Click `Finish` to add a reference to `System.Data.Entity.dll`, which contains the EF classes, generate the XML and partial class files, and open the EDM Designer window. The Designer displays the two `EntityType`s, `Orders` and `Order_Details`, and a single `AssociationSet`, `FK_Order_Details_Orders`. The Model Browser pane shows nodes and subnodes for the `modelName` EDM and `modelName`.

## Chapter 9: Raising the Level of Data Abstraction

Store database. The Mapping Details pane displays the default mapping between the table's columns and the EntityType's properties. (see Figure 9-6).

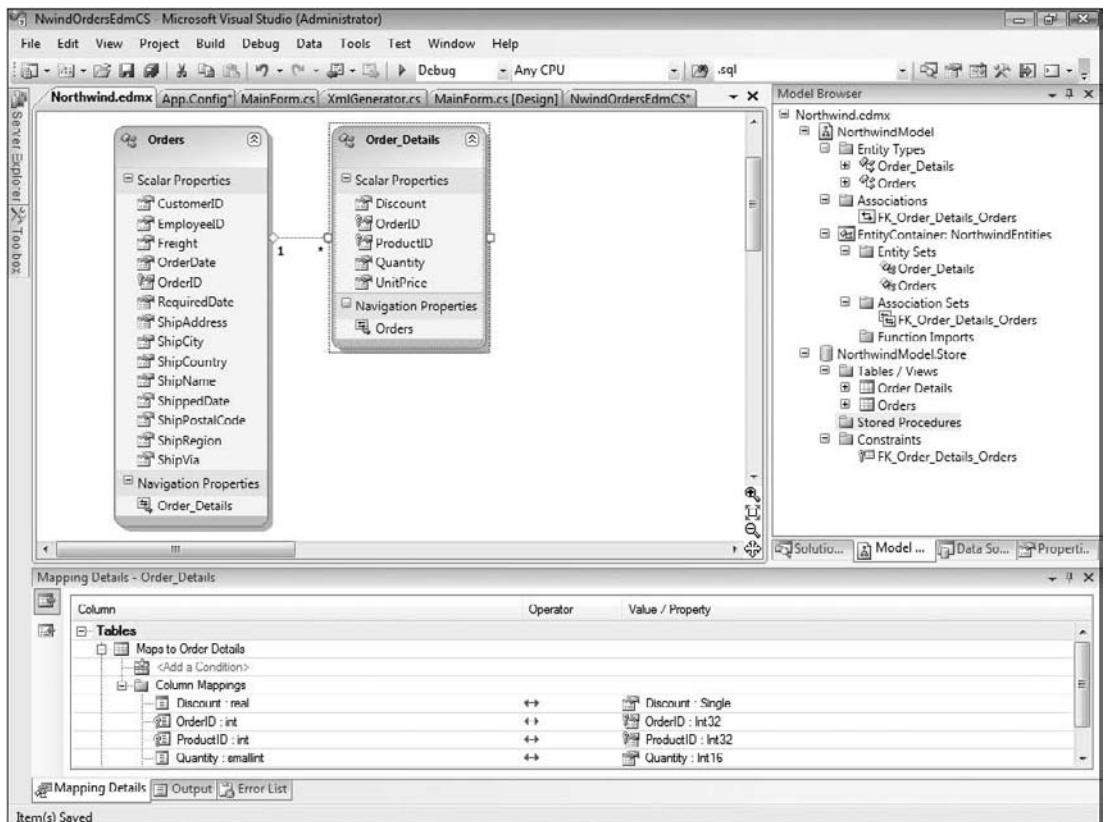


Figure 9-6

### **Adding, Updating, and Deleting the Model's Objects**

If you want to add new database objects or update the mapping layers to reflect changes to the schema, right-click the Model Browser and choose Update Model from Database to open a modified version of the Wizard's Choose Your Database Objects dialog that contains Add, Update, and Delete tabs for the three database object types (see Figure 9-7).

## Part IV: Introducing the ADO.NET Entity Framework

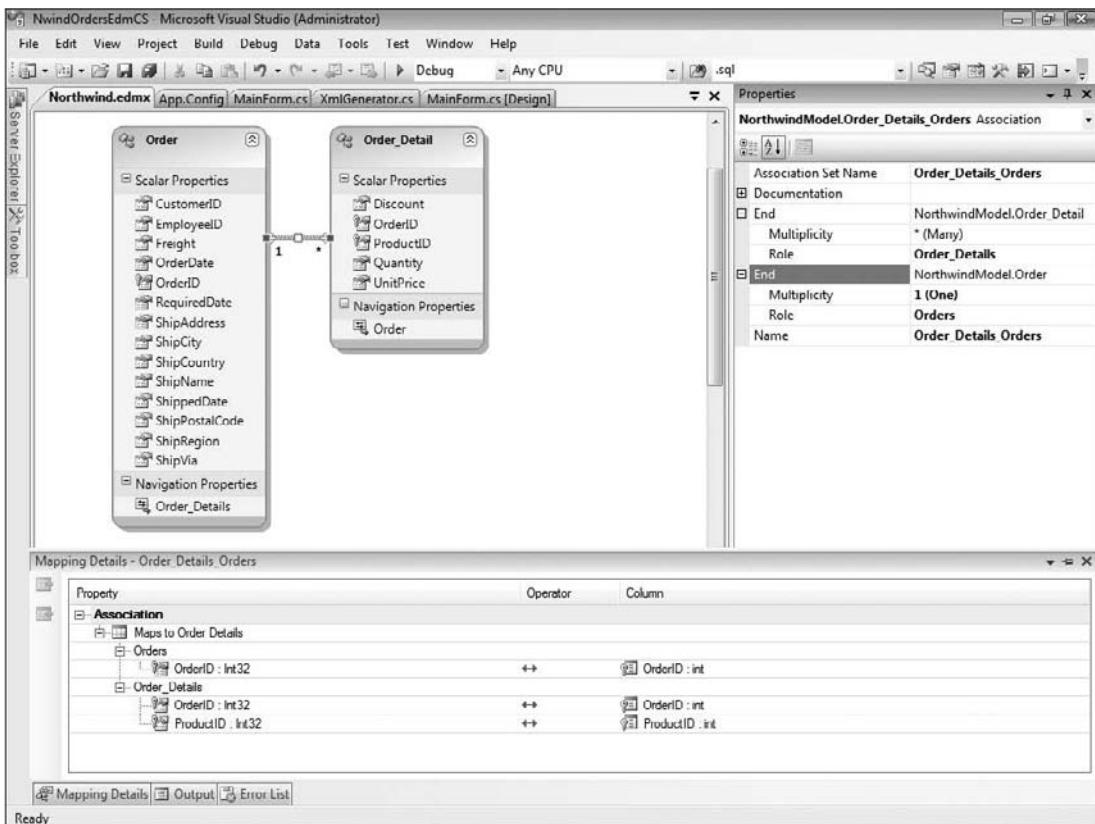


Figure 9-7

You can select to add any object that isn't present in your model; all objects in the model are updated when you click Finish. To delete an `EntityType` from the model, select it and then press Delete and F5 to rebuild the model. (You can't mark an object for deletion in the dialog.)

### **Editing EntityType and AssociationSet Properties**

Commonly used property values of object layer elements can be edited in the EDM Designer, either by editing text in the Entity tool or the elements' properties sheet.

By default, the wizard names `EntityType`s and `EntitySets` from the store's table name, which commonly is plural. In this case, you can singularize the `EntityType`'s `Name` property in the Entity tool's text box. If the table name is singular, right-click the Entity tool, choose Properties to open the properties sheet, and pluralize the `Entity Set Name` property.

*If you singularize the EntityType's Name property, the designer changes the EntitySet's Name property to NameSet, such as OrderSet and Order\_DetailSet for this example. Unlike LINQ to SQL, you can't turn off EF's arbitrary EntitySet naming convention. The ADO.NET team has promised to make this "feature" optional in EF v2, but you must manually pluralize EntitySet names in EF v1.*

## Chapter 9: Raising the Level of Data Abstraction

Similarly, the wizard names Navigation Properties without regard to the Multiplicity property value of the AssociationSet's Ends. It's preferable that AssociationSet names reflect the Multiplicity value of their Ends and that referential "noise" such as the FK prefix be removed (see Figure 9-8).

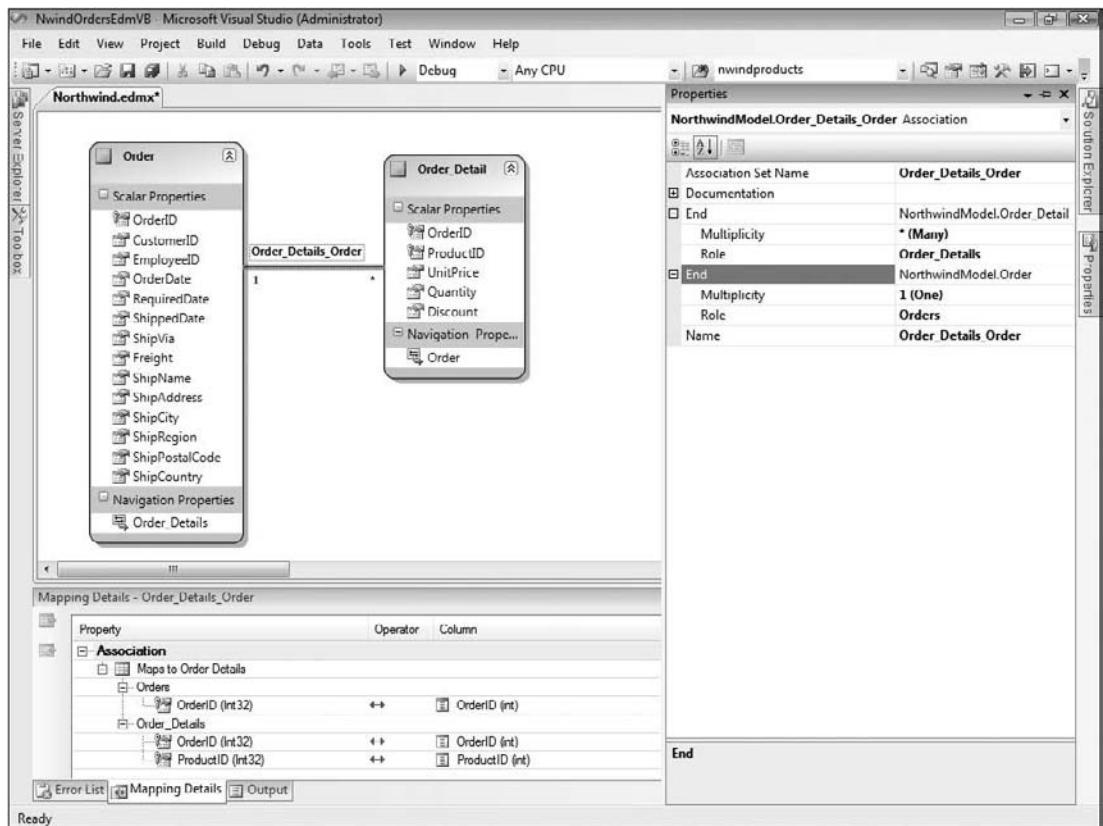


Figure 9-8

In Figure 9-8, the EntityType names and Orders Navigation Property name have been singularized, and the FK\_prefix has been removed from the AssociationSet name. The Association mapping information appears in the mapping pane when you select an AssociationSet. Foreign-key values for the CustomerID, EmployeeID, and ShipperID properties are present because the target EntitySets and AssociationSets are missing from the EntityContainer.

You might also need to change the Multiplicity value from 0...1 to 1 of Ends that represent foreign-key fields and that have referential integrity enforced but allow NULL values. (Most Northwind foreign keys, except OrderID and ProductID, which are members of the Order Details table's primary key, have this referential conflict.)

### Analyzing the ModelName.edmx File's Sections

During most of EF's beta period, *ModelName.ssdl*, *ModelName.csdl*, and *ModelName.msl* were ordinary XML files stored in the project's ... \Bin\Debug or ... \Bin\Release folders. Shortly before the release of EF in VS 2008 SP1, the three files were moved from an accessible folder to hidden resources and the *ModelName.edmx* file gained corresponding `<edmx:StorageModels>`, `<edmx:ConceptualModels>`, and `<edmx:mappings>` groups in the `<edmx:Runtime>` section. The following sections describe the content of these three groups.

#### Scanning the StorageModels Group

The `<edmx:StorageModels>` group represents the schema for the metadata of selected database objects in Store Schema Definition Language (SSDL). Following is the content of the `<edmx:StorageModels>` group of *Northwind.edmx* with Orders and Order Details tables:

#### XML: SSDL Content - Store Schema Definition Language (SSDL)

```
<!-- SSDL content -->
<edmx:StorageModels>
  <Schema Namespace="NorthwindModel.Store" Alias="Self"
    Provider="System.Data.SqlClient" ProviderManifestToken="2005" xmlns:store=
    "http://schemas.microsoft.com/ado/2007/12/edm/EntityStoreSchemaGenerator"
    xmlns="http://schemas.microsoft.com/ado/2006/04/edm/ssdl">
    <EntityContainer Name="NorthwindModelStoreContainer">
      <EntityType Name="Order Details" EntityType=
        "NorthwindModel.Store.Order Details" store:Type="Tables" Schema="dbo" />
      <EntityType Name="Orders" EntityType="NorthwindModel.Store.Orders"
        store:Type="Tables" Schema="dbo" />
      <AssociationSet Name="FK_Order_Details_Orders"
        Association="NorthwindModel.Store.FK_Order_Details_Orders">
        <End Role="Orders" EntitySet="Orders" />
        <End Role="Order Details" EntitySet="Order Details" />
      </AssociationSet>
    </EntityContainer>
    <EntityType Name="Order Details">
      <Key>
        <PropertyRef Name="OrderID" />
        <PropertyRef Name="ProductID" />
      </Key>
      <Property Name="Discount" Type="real" Nullable="false" />
      <Property Name="OrderID" Type="int" Nullable="false" />
      <Property Name="ProductID" Type="int" Nullable="false" />
      <Property Name="Quantity" Type="smallint" Nullable="false" />
      <Property Name="UnitPrice" Type="money" Nullable="false" />
    </EntityType>
    <EntityType Name="Orders">
      <Key>
        <PropertyRef Name="OrderID" />
      </Key>
      <Property Name="CustomerID" Type="nchar" MaxLength="5" />
      <Property Name="EmployeeID" Type="int" />
      <Property Name="Freight" Type="money" />
      <Property Name="OrderDate" Type="datetime" />
      <Property Name="OrderID" Type="int" Nullable="false" />
```

```
        StoreGeneratedPattern="Identity" />
<Property Name="RequiredDate" Type="datetime" />
<Property Name="ShipAddress" Type="nvarchar" MaxLength="60" />
<Property Name="ShipCity" Type="nvarchar" MaxLength="15" />
<Property Name="ShipCountry" Type="nvarchar" MaxLength="15" />
<Property Name="ShipName" Type="nvarchar" MaxLength="40" />
<Property Name="ShippedDate" Type="datetime" />
<Property Name="ShipPostalCode" Type="nvarchar" MaxLength="10" />
<Property Name="ShipRegion" Type="nvarchar" MaxLength="15" />
<Property Name="ShipVia" Type="int" />
</EntityType>
<Association Name="FK_Order_Details_Orders">
    <End Role="Orders" Type="NorthwindModel.Store.Orders" Multiplicity="1" />
    <End Role="Order Details" Type="NorthwindModel.Store.Order Details"
        Multiplicity="*" />
    <ReferentialConstraint>
        <Principal Role="Orders">
            <PropertyRef Name="OrderID" />
        </Principal>
        <Dependent Role="Order Details">
            <PropertyRef Name="OrderID" />
        </Dependent>
    </ReferentialConstraint>
</Association>
</Schema>
</edmx:StorageModels>
```

Most element names and values are self-explanatory. EntityContainer is a synonym for database but the value of that element is dbo, the name of the tables' schema. If you add tables from multiple SQL Server schemas, the entity container will adopt one of the schema names and add a Schema="SchemaName" attribute to EntitySet elements for tables from other schema(s).

*If an EntityType doesn't have a readily identifiable primary key, the wizard generates a DefiningQuery to create a read-only view of the entity. If you verify that a column or columns can uniquely define an instance, you can edit the SSDL to reflect the change. However, specifying a key or composite key in the database table definition is a better approach because edits to the SSDL are overwritten when you update the mapping files from the database.*

## **Examining the ConceptualModels Group**

The <edmx:ConceptualModels> group represents, in Conceptual Schema Definition Language (CSDL), the schema for the EDM of objects mapped to and from the store layer. Following are brief descriptions of the CSDL's structure:

- ❑ The Model Namespace text box of the last EDM Wizard dialog assigns CSDL's top-level <Schema> element's Namespace attribute value.
- ❑ The Entity Connection Settings text box assigns the <EntityContainer> element's Name attribute value.
- ❑ <EntitySet> elements have Name and EntityType attributes.
- ❑ <AssociationSet> elements have Name and Association attributes, as well as <End> child elements with Role and EntitySet attributes.

## Part IV: Introducing the ADO.NET Entity Framework

---

- ❑ <EntityType> elements have a Name attribute and <Key>, <Property> and <NavigationProperty> child elements. DataType attributes of <Property> elements can be CLR scalar or complex types; complex types (more commonly called *value types*) consist of one or more scalar or complex properties, which you access like other scalar properties, and aren't identified by a key value. <Property> elements can have optional attributes, called *named facets*, to specify additional constraints — typically nullability and maximum length.
- ❑ <Association> elements have a Name attribute, and <End> and <ReferentialConstraint> elements. The latter element appears to introduce a store-related dependency into an otherwise persistence-ignorant conceptual layer.

An example of a complex or value property is a <USAddress> class that might have four or five scalar properties, such as <StreetAddress>, <PostOfficeBox>, <City>, <State>, and <Zip>; <CdnAddress> might substitute <Province> and <PostalCode> for Canadian addresses. Both would be subtypes of the <Address> class.

Following is the CSDL content mapped from the SSDL content of Northwind.edmx by the C-S Mapping content, which the next section covers:

### XML: CSDL Content - Conceptual Schema Definition Language (CSDL)

```
<!-- CSDL content -->
<edmx:ConceptualModels>
  <Schema Namespace="NorthwindModel" Alias="Self"
    xmlns="http://schemas.microsoft.com/ado/2006/04/edm">
    <EntityContainer Name="NorthwindEntities">
      <EntityType Name="Order_Details" EntityType="NorthwindModel.Order_Detail" />
      <EntityType Name="Orders" EntityType="NorthwindModel.Order" />
      <AssociationSet Name="Order_Details_Orders"
        Association="NorthwindModel.Order_Details_Orders">
        <End Role="Orders" EntitySet="Orders" />
        <End Role="Order_Details" EntitySet="Order_Details" />
      </AssociationSet>
    </EntityContainer>
    <EntityType Name="Order_Detail">
      <Key>
        <PropertyRef Name="OrderID" />
        <PropertyRef Name="ProductID" />
      </Key>
      <Property Name="Discount" Type="Single" Nullable="false" />
      <Property Name="OrderID" Type="Int32" Nullable="false" />
      <Property Name="ProductID" Type="Int32" Nullable="false" />
      <Property Name="Quantity" Type="Int16" Nullable="false" />
      <Property Name="UnitPrice" Type="Decimal" Nullable="false"
        Precision="19" Scale="4" />
      <NavigationProperty Name="Order"
        Relationship="NorthwindModel.Order_Details_Orders"
        FromRole="Order_Details" ToRole="Orders" />
    </EntityType>
    <EntityType Name="Order">
      <Key>
        <PropertyRef Name="OrderID" />
      </Key>
      <Property Name="CustomerID" Type="String" MaxLength="5" Unicode="true"
```

```
    FixedLength="true" />
<Property Name="EmployeeID" Type="Int32" />
<Property Name="Freight" Type="Decimal" Precision="19" Scale="4" />
<Property Name="OrderDate" Type="DateTime" />
<Property Name="OrderID" Type="Int32" Nullable="false" />
<Property Name="RequiredDate" Type="DateTime" />
<Property Name="ShipAddress" Type="String" MaxLength="60" Unicode="true"
    FixedLength="false" />
<Property Name="ShipCity" Type="String" MaxLength="15" Unicode="true"
    FixedLength="false" />
<Property Name="ShipCountry" Type="String" MaxLength="15" Unicode="true"
    FixedLength="false" />
<Property Name="ShipName" Type="String" MaxLength="40" Unicode="true"
    FixedLength="false" />
<Property Name="ShippedDate" Type="DateTime" />
<Property Name="ShipPostalCode" Type="String" MaxLength="10" Unicode="true"
    FixedLength="false" />
<Property Name="ShipRegion" Type="String" MaxLength="15" Unicode="true"
    FixedLength="false" />
<Property Name="ShipVia" Type="Int32" />
<NavigationProperty Name="Order_Details"
    Relationship="NorthwindModel.Order_Details_Orders"
    FromRole="Orders" ToRole="Order_Details" />
</EntityType>
<Association Name="Order_Details_Orders">
    <End Role="Orders" Type="NorthwindModel.Order" Multiplicity="1" />
    <End Role="Order_Details" Type="NorthwindModel.Order_Detail"
        Multiplicity="*" />
    <ReferentialConstraint>
        <Principal Role="Orders">
            <PropertyRef Name="OrderID" />
        </Principal>
        <Dependent Role="Order_Details">
            <PropertyRef Name="OrderID" />
        </Dependent>
    </ReferentialConstraint>
</Association>
</Schema>
</edmx:ConceptualModels>
```

A relation table that consists solely of pairs of foreign-key values and has no payload (other columns), map to a many:many (m:n, \*:\*) Association, not an EntityType, in the ConceptualModels group. This differs from the behavior of LINQ to SQL, which doesn't support many:many associations with no payload.

### Tracing the Mappings Group

The `<edmx:ConceptualModels>` group represents in Mapping Specification Language (MSL) the connection of types declared in the `<edmx:ConceptualModels>` group and database metadata that's defined in the corresponding `<edmx:StorageModels>` group. The `StorageEntityContainer` and `CdmEntityContainer` attributes identify the database schema name and CSDL `<EntityContainer>` element's `Name` attribute value.

## Part IV: Introducing the ADO.NET Entity Framework

---

Following is the content of the `<edmx:Mappings>` group for mapping between `<edmx:StorageModels>` and `<edmx:ConceptualModels>` groups:

```
<!-- C-S mapping content -->
<edmx:Mappings>
  <Mapping Space="C-S" xmlns="urn:schemas-microsoft-com:windows:storage:mapping:CS">
    <EntityContainerMapping StorageEntityContainer="NorthwindModelStoreContainer"
      CdmEntityContainer="NorthwindEntities">
      <EntityTypeMapping TypeName="IsTypeOf(NorthwindModel.Order_Detail)">
        <MappingFragment StoreEntitySet="Order Details">
          <ScalarProperty Name="Discount" ColumnName="Discount" />
          <ScalarProperty Name="OrderID" ColumnName="OrderID" />
          <ScalarProperty Name="ProductID" ColumnName="ProductID" />
          <ScalarProperty Name="Quantity" ColumnName="Quantity" />
          <ScalarProperty Name="UnitPrice" ColumnName="UnitPrice" />
        </MappingFragment>
      </EntityTypeMapping>
    </EntityTypeMapping>
    <EntityTypeMapping TypeName="IsTypeOf(NorthwindModel.Order)">
      <MappingFragment StoreEntitySet="Orders">
        <ScalarProperty Name="CustomerID" ColumnName="CustomerID" />
        <ScalarProperty Name="EmployeeID" ColumnName="EmployeeID" />
        <ScalarProperty Name="Freight" ColumnName="Freight" />
        <ScalarProperty Name="OrderDate" ColumnName="OrderDate" />
        <ScalarProperty Name="OrderID" ColumnName="OrderID" />
        <ScalarProperty Name="RequiredDate" ColumnName="RequiredDate" />
        <ScalarProperty Name="ShipAddress" ColumnName="ShipAddress" />
        <ScalarProperty Name="ShipCity" ColumnName="ShipCity" />
        <ScalarProperty Name="ShipCountry" ColumnName="ShipCountry" />
        <ScalarProperty Name="ShipName" ColumnName="ShipName" />
        <ScalarProperty Name="ShippedDate" ColumnName="ShippedDate" />
        <ScalarProperty Name="ShipPostalCode" ColumnName="ShipPostalCode" />
        <ScalarProperty Name="ShipRegion" ColumnName="ShipRegion" />
        <ScalarProperty Name="ShipVia" ColumnName="ShipVia" />
      </MappingFragment>
    </EntityTypeMapping>
  </EntityTypeMapping>
  <AssociationSetMapping Name="Order_Details_Orders"
    TypeName="NorthwindModel.Order_Details_Orders">
    <StoreEntitySet="Order Details">
      <EndProperty Name="Orders">
        <ScalarProperty Name="OrderID" ColumnName="OrderID" />
      </EndProperty>
      <EndProperty Name="Order_Details">
        <ScalarProperty Name="OrderID" ColumnName="OrderID" />
        <ScalarProperty Name="ProductID" ColumnName="ProductID" />
      </EndProperty>
    </AssociationSetMapping>
  </EntityContainerMapping>
</Mapping>
</edmx:Mappings>
```

These three mapping files represent a simple, 1:1 mapping of `EntityType`s to database tables, which is the sweet spot for LINQ to SQL. Chapter 10 demonstrates more complex scenarios that involve mapping multiple tables to a single `EntityType` and vice versa, and other mapping esoterica. Chapter 13 covers working with complex types.

# Working with the Entity Client, Entity SQL and Client Views

The `EntityClient` ADO.NET data provider (`System.Data.EntityClient` namespace) enables access to the EDM's conceptual `EntityContainer` by executing statements written in the Entity SQL dialect of ANSI SQL. The simplest implementation delivers client views that support hierarchical structures, nested, and polymorphic results, and complex data types from `EntityConnections` and `EntityCommands` that produce the Canonical Query Tree. An underlying storage-specific ADO.NET provider translates the CQT to `DbCommand.CommandText` in the RDBMS's SQL dialect and the RDBMS processes the query. The `EntityCommand.ExecuteReader()` method processes the query resultset and returns the client view as an untyped `EntityDataReader` to the `EntityClient`. The `EntityDataReader` implements `IExtendedDataRecord` to process structured resultsets.

A `DbDataReader` that implements `IExtendedDataRecord` can return any of the following types in its columns:

- A scalar type, such as `Int32`, `String`, or `DateTime`
- A nested `DbDataRecord` type containing an entity instance
- A nested `DbDataReader` type containing an entity collection
- A nested `EntityKey` type with one or more `EntityKeyMembers`, each of which has a `Key (String)` and `Value (Object)` property

Code in the `DbDataReader.Read()` loop tests the type with `if (typRecord == typeof(DbDataRecord)) ... / If typRec Is GetType(DbDataRecord) Then ...` structures and processes the column accordingly. The next section describes procedures that generate text and XML representations of `EntityQuery` resultsets with all but a nested `EntityKey` type.

EF doesn't have a class for or assign a proper name to queries generated with the `EntityClient`'s classes that correspond to ADO.NET's `DbConnection`, `DbCommand`, and `DbDataReader` objects, so this book calls them `EntityQueries`. Other query types, such as the `ObjectQuery` and LINQ to Entities queries build on the `EntityDataReader`, so executing `EntityQuery` has the least overhead of the three EF query types. This chapter includes sections with examples and brief descriptions `ObjectQuery` and LINQ to Entities queries, which Chapter 11 and Chapter 12 describe in detail.

### Writing EntityQueries in Entity SQL

Entity SQL is deliberately similar to ANSI SQL-92 with a few missing keywords and constructs, no self-contained functions, and many entity-specific additions to the grammar. Here are some of the most common differences from SQL-92 that you'll notice in entry-level Entity SQL queries:

- ❑ EntityQueries execute against `EntityContainer.EntityCollection` objects, `NorthwindEntitiesCS.Orders` and `NorthwindEntitiesVB.Orders` for the `NwindOrdersEdmCS` and VB sample projects.
- ❑ All references to entities, as well as GROUP BY keys, require an alias prefix as in `SELECT o.OrderID, o.CustomerID, o.EmployeeID ... FROM NorthwindEntitiesCS.Orders AS o.`
- ❑ The \* wildcard isn't supported for projections; to return a projection, you must use `SELECT VALUE o` FROM `NorthwindEntitiesCS.Orders AS o` or `EntityAlias.PropertyName` as in the preceding example. `SELECT VALUE` can only project a single item.
- ❑ An `EntityAlias` without a `PropertyName`, as in `SELECT o` FROM `NorthwindEntitiesCS.Orders AS o` returns a nested `DbDataRecord` for each table row instead of columns of scalar values.
- ❑ Navigation properties replace INNER JOIN clauses across relationships with `EntityAlias.PropertyName.NavigationProperty` syntax, as in `SELECT o, o.Order_Details` FROM `NorthwindEntitiesCS.Orders AS o`, which returns a nested `DbDataRecord` containing an `Order` entity and a `DbDataReader` containing an associated `Order_Detail` collection for each row.
- ❑ The NAVIGATE keyword provides an alternative method for navigating associations.
- ❑ JOINS are supported for combining entity instances without navigation properties.
- ❑ IN and EXISTS subqueries and UNION, INTERSECT and EXCEPT set operators support collections.
- ❑ TOP, SKIP, and LIMIT sub-clauses handle paging.
- ❑ `DbCommand.Parameters(n).Name` values won't accept an @ symbol as a prefix. Valid parameter names must begin with a letter followed by letters, digits, or underscores.

Online help for the Entity Framework includes "Entity SQL Overview" and "Entity SQL Reference" chapters with terse descriptions and cryptic examples of Entity SQL syntax.

### Executing Entity SQL Queries as Client Views

The two `NwindOrdersEDM` sample projects let you compare execution of EntityQueries and conventional ADO.NET queries against a subset of the Northwind Orders and Order Details tables. Here's the code for the EntityQuery button's Click event handler:

### C# 3.0

```
private void btnEntityQuery_Click(object sender, System.EventArgs e)
{
    //Execute an EntityCommand to retrieve US orders and line items
    StringBuilder sbData = new StringBuilder();
    Stopwatch swTimer = new Stopwatch();
    string esqlQuery = null;
    string sqlQuery = null;
    swTimer.Start();

    using (EntityConnection ecnnNwind =
        new EntityConnection("name=NorthwindEntities"))
    {
        // Parameterized Entity SQL query
        esqlQuery = "SELECT o FROM NorthwindEntities.Orders AS o " +
            "WHERE o.ShipCountry = @Country ORDER BY o.OrderDate DESC";
        if (chkDetails.Checked)
        {
            // Add a nested DbDataReader for Order_Details
            esqlQuery = esqlQuery.Replace("SELECT o",
                "SELECT o, o.Order_Details");
        }

        using (EntityCommand ecmdNwind = ecnnNwind.CreateCommand())
        {
            ecmdNwind.CommandText = esqlQuery;
            // Add the parameter (Notice the missing @ prefix)
            ecmdNwind.Parameters.AddWithValue("Country", "USA");
            ecnnNwind.Open();
            using (EntityDataReader edrReader =
                ecmdNwind.ExecuteReader(CommandBehavior.SequentialAccess))
            {
                if (chkGenerateXML.Checked)
                {
                    // Generate XML document
                    _XmlDoc = new XmlDocument();
                    _XmlCommand = _XmlDoc.CreateElement("Command");
                    _XmlDoc.AppendChild(_XmlCommand);
                    VisitReader(edrReader);
                }
                else
                {
                    // Generate plain text version
                    WriteReaderText(edrReader, sbData);
                }
            }
            // Store-specific SQL sent to RDBMS query engine
            sqlQuery = ecmdNwind.ToString();
        }
    }
    txtTime.Text = (swTimer.ElapsedTicks / 10000).ToString("##0.000");
    if (chkGenerateXML.Checked)
```

## Part IV: Introducing the ADO.NET Entity Framework

---

```
{  
    // Create an XML file for IE and display content in text box  
    CreateAndDisplayXml();  
}  
else  
{  
    // Display the raw text data  
    txtData.Text = sbData.ToString();  
}  
// Display Entity SQL and store-specific SQL  
txtData.Text += "\r\nEntity SQL: " +  
    esqlQuery + "\r\n\r\nT-SQL: " + sqlQuery;  
}
```

*Specific code lines for the Entity SQL Query are highlighted. The VisitReader() and WriteReaderText() procedures generate XML and text documents, respectively, from the DbDataReader. WriteReaderText() provides an example of processing nested DbDataRecord, DbDataReader and scalar types.*

### VB 9.0

```
Private Sub btnEntityQuery_Click(ByVal sender As System.Object, _  
    ByVal e As System.EventArgs) Handles btnEntityQuery.Click  
    'Execute an EntityCommand to retrieve US orders and line items  
    Dim sbData As New StringBuilder  
    Dim swTimer As New Stopwatch  
    Dim esqlQuery As String  
    Dim sqlQuery As String  
    swTimer.Start()  
  
    Using ecnnNwind As New EntityConnection("name=NorthwindEntities")  
        ' Parameterized Entity SQL query  
        esqlQuery = "SELECT o FROM NorthwindEntities.Orders AS o " & _  
            "WHERE o.ShipCountry = @Country ORDER BY o.OrderDate DESC"  
        If chkDetails.Checked Then  
            ' Add a nested DbDataReader for Order_Details  
            esqlQuery = esqlQuery.Replace("SELECT o", "SELECT o, o.Order_Details")  
        End If  
        Using ecmdNwind As EntityCommand = ecnnNwind.CreateCommand()  
            ecmdNwind.CommandText = esqlQuery  
            ' Add the parameter (Notice the missing @ prefix)  
            ecmdNwind.Parameters.AddWithValue("Country", "USA")  
            ecnnNwind.Open()  
            Using edrReader As EntityDataReader = _  
                ecmdNwind.ExecuteReader(CommandBehavior.SequentialAccess)  
                If chkGenerateXML.Checked Then  
                    ' Generate XML document  
                    _XmlDoc = New XmlDocument()  
                    _XmlCommand = _XmlDoc.CreateElement("Command")  
                    _XmlDoc.AppendChild(_XmlCommand)  
                    VisitReader(edrReader)  
                End If  
            End Using  
        End Using  
    End Using
```

```
        Else
            ' Generate plain text version
            WriteReaderText(edrReader, sbData)
        End If
    End Using
    ' Store-specific SQL sent to RDBMS query engine
    sqlQuery = ecmdNwind.ToString()
End Using
txtTime.Text = (swTimer.ElapsedTicks / 10000).ToString("##0.000")
If chkGenerateXML.Checked Then
    ' Create an XML file for IE and display content in text box
    CreateAndDisplayXml()
Else
    ' Display the raw text data
    txtData.Text = sbData.ToString()
End If
' Display Entity SQL and store-specific SQL
txtData.Text &= vbCrLf & vbCrLf & "Entity SQL: " & _
    esqlQuery & vbCrLf & vbCrLf & "T-SQL: " & sqlQuery
End Sub
```

Figure 9-9 shows the UI of the NwindOrdersEdmCS.sln project displaying the start of an XML document with parent records for Order entities and child groups for records from the Order\_Details association.

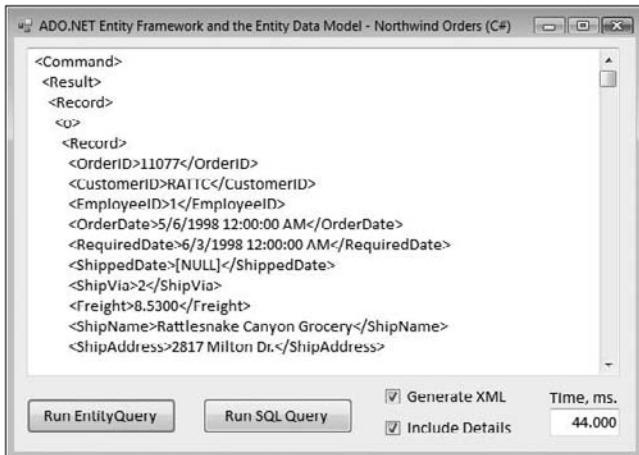


Figure 9-9

The following table lists the time in milliseconds to execute with the NwindOrdersEdmCS.sln (or VB) project cached Entity SQL and T-SQL queries that return the same amount of data. It's reasonable for Entity SQL queries to be slower to execute than T-SQL Queries, because T-SQL doesn't require creating and translating the CQT or handling hierarchical resultsets.

## Part IV: Introducing the ADO.NET Entity Framework

---

Options Selected	Entity SQL	T-SQL
Text without Order Details	10 ms.	7 ms.
XML without Order Details	17 ms.	11 ms.
Text with Order Details	27 ms.	17 ms.
XML with Order Details	42 ms.	21 ms.

Following is the Entity SQL statement and the T-SQL batch generated by the EntityClient's CQT translated to T-SQL by SqlClient:

### **Entity SQL from ecmdNwind.CommandText**

```
SELECT o, o.Order_Details FROM NorthwindEntities.Orders AS o  
WHERE o.ShipCountry = @Country ORDER BY o.OrderDate DESC
```

### **T-SQL from ToTraceString()**

```
SELECT  
[Project1].[OrderID] AS [OrderID],  
[Project1].[CustomerID] AS [CustomerID],  
[Project1].[EmployeeID] AS [EmployeeID],  
[Project1].[OrderDate] AS [OrderDate],  
[Project1].[RequiredDate] AS [RequiredDate],  
[Project1].[ShippedDate] AS [ShippedDate],  
[Project1].[ShipVia] AS [ShipVia],  
[Project1].[Freight] AS [Freight],  
[Project1].[ShipName] AS [ShipName],  
[Project1].[ShipAddress] AS [ShipAddress],  
[Project1].[ShipCity] AS [ShipCity],  
[Project1].[ShipRegion] AS [ShipRegion],  
[Project1].[ShipPostalCode] AS [ShipPostalCode],  
[Project1].[ShipCountry] AS [ShipCountry],  
[Project1].[C1] AS [C1],  
[Project1].[C2] AS [C2],  
[Project1].[OrderID1] AS [OrderID1],  
[Project1].[ProductID] AS [ProductID],  
[Project1].[UnitPrice] AS [UnitPrice],  
[Project1].[Quantity] AS [Quantity],  
[Project1].[Discount] AS [Discount]  
FROM ( SELECT  
[Extent1].[OrderID] AS [OrderID],  
[Extent1].[CustomerID] AS [CustomerID],  
[Extent1].[EmployeeID] AS [EmployeeID],  
[Extent1].[OrderDate] AS [OrderDate],  
[Extent1].[RequiredDate] AS [RequiredDate],  
[Extent1].[ShippedDate] AS [ShippedDate],  
[Extent1].[ShipVia] AS [ShipVia],  
[Extent1].[Freight] AS [Freight],  
[Extent1].[ShipName] AS [ShipName],  
[Extent1].[ShipAddress] AS [ShipAddress],  
[Extent1].[ShipCity] AS [ShipCity],
```

```
[Extent1].[ShipRegion] AS [ShipRegion],  
[Extent1].[ShipPostalCode] AS [ShipPostalCode],  
[Extent1].[ShipCountry] AS [ShipCountry],  
1 AS [C1],  
[Extent2].[OrderID] AS [OrderID1],  
[Extent2].[ProductID] AS [ProductID],  
[Extent2].[UnitPrice] AS [UnitPrice],  
[Extent2].[Quantity] AS [Quantity],  
[Extent2].[Discount] AS [Discount],  
CASE WHEN ([Extent2].[OrderID] IS NULL) THEN CAST(NULL AS int)  
     ELSE 1 END AS [C2]  
FROM  [dbo].[Orders] AS [Extent1]  
LEFT OUTER JOIN [dbo].[Order Details] AS [Extent2]  
    ON [Extent1].[OrderID] = [Extent2].[OrderID]  
   WHERE [Extent1].[ShipCountry] = @Country  
) AS [Project1]  
ORDER BY [Project1].[OrderDate] DESC, [Project1].[OrderID] ASC, [Project1].[C2] ASC
```

Executing the preceding batch in SQL Server Management Studio sends an extraordinarily long statement to generate a conventional LEFT OUTER JOIN that's the same as ordinarily generated by:

### T-SQL

```
SELECT o.* , d.*  
FROM Orders AS o LEFT OUTER JOIN [Order Details] AS d ON d.OrderID = o.OrderID  
WHERE ShipCountry = 'USA'  
ORDER BY o.OrderID DESC
```

The ObjectQuery type's native language is Entity SQL, so you'll learn more about Entity SQL in the next few sections and Chapter 10.

## Taking Advantage of Object Services

The Object Services layer — implemented by classes in the `System.Data.Objects` and `System.Data.Objects.DataClasses` namespaces — is a set of components that implements EF's O/RM features. Object Services include the following capabilities:

- ❑ Create `ObjectContext` instances that incorporate `EntityConnection`, `MetadataWorkspace`, and `ObjectStateManager` objects.
- ❑ Derive `EntityContainer` classes from `ObjectContext` and entity classes from the `EntityObject` class.
- ❑ Write and execute `ObjectQuery` instances using Entity SQL statements, query builder methods, or LINQ to Entities expressions or method calls.
- ❑ Return strongly typed instances of entity types as `IEnumerable<EntityType>` sequences.
- ❑ Identify entity instances in the cache.
- ❑ Insert, update and delete entity instances in the cache.

## Part IV: Introducing the ADO.NET Entity Framework

---

- Track cached data changes for individual or batch updates within transactions by invoking the `ObjectContext.SaveChanges()` method.
- Manage concurrency conflicts.
- Bind entity collections to data-aware Windows and Web controls, including the ASP.NET `EntityDataSource`.

LINQ to SQL's `DataContext` object offers a feature set that's similar to Object Services, except for multilingual queries. Object Services features are, on the whole, more comprehensive and configurable than corresponding LINQ to SQL capabilities.

The following sections provide a brief introduction to the `ObjectContext`, `ObjectQuery`, and using query builder methods as a substitute for Entity SQL. The sample projects for Object Services and LINQ to Entities topics are `NwindObjectServicesCS.sln` and `NwindObjectServicesVB.sln` in the `\WROX\ADONET\Chapter09\CS` and `...\VB` folders.

### **Working with `ObjectContext`**

`ObjectContext`'s most important properties are the `EntityCollection`'s `EntitySets` which `ObjectContext` exposes as `ObjectQuery<EntityType>`s, `MetadataWorkspace`, and `ObjectStateManager`.

### **Creating an Object Context and `ObjectQuery`**

The `ModelName.Designer.cs` or `ModelName.Designer.vb` file you create with the EDM Wizard contains the top-level class with the name you assigned to the connection string, `nwEntities` for this example, which inherits from `System.Data.Objects.ObjectContext`. `nwEntities` has a parameterless constructor that supplies the connection string saved in `App.config` or `Web.config`; alternatively, you can pass a connection string or an `EntityConnection` instance to overloads.

Ordinarily, you start by creating a new `NorthwindEntities` instance with an instruction such as this:

```
NorthwindEntities ctxNwind = new NorthwindEntities();
```

or

```
Dim ctxNwind = New NorthwindEntities()
```

*Both versions of the project use the default EntityModel and EntityContainer names.*

You then invoke the `ctxNwind.CreateQuery()` method to create and prepare to execute `ObjectQuery<EntityType>` against the data store. Alternatively, you can initialize a new `ObjectQuery<EntityType>` and assign it to `ctxNwind`. Like LINQ queries, `ObjectQuery(Of T)`'s defer execution until code requires materializing the sequence. `ObjectQuery(Of T)` implements the `IOrderedQueryable(Of T)`, `IQueryable(Of T)`, `IEnumerable(Of T)`, `IOrderedQueryable`, `IQueryable`, `IEnumerable`, and `IListSource` interfaces, so `ObjectQuery(Of T)`'s are composable and bindable.

*The later "Writing ObjectQueries with Entity SQL," "Composing ObjectQueries with Query Builder Methods," and "Using the LINQ to Entities Provider" sections describe the three languages you can use to write `ObjectQuery(Of T)`'s and show you how to use them as data sources for data-aware controls.*

## MetadataWorkspace

Following are four of the more important metadata collections that the `MetadataWorkspace` contains:

- ❑ `ObjectItemCollection` of metadata for the object model (`OSpace`)
- ❑ `EdmItemCollection` of metadata for the conceptual model (`CSDL, CSpace`)
- ❑ `StoreItemCollection` of metadata for the storage model (`SSDL, SSpace`)
- ❑ `StorageMappingItemCollection` of metadata for mapping between the storage and conceptual model (`MSL, CSSpace`)

You can iterate these collections with code such as the following to gain additional insight into the metadata produced by the mapping files:

### C# 3.0

```
private void SaveWorkspaceMetadata()
{
    MetadataWorkspace nwMdws = ctxNwind.MetadataWorkspace;
    ReadOnlyCollection<EdmType> nwTypes =
        nwMdws.GetItems<EdmType>(DataSpace.OSpace);
    sbType.Append("ObjectItemCollection (OSpace)\r\n");
    foreach (EdmType nwType in nwTypes)
    {
        sbType.Append("ObjectItem: " + nwType.GetType().FullName + ": " +
            nwType.FullName + "\r\n");
        foreach (MetadataProperty nwProp in nwType.MetadataProperties)
        {
            sbType.Append("Property: " + nwProp.Name + ((nwProp.Value != null)
                ? ":" + nwProp.Value.ToString() : "") + "\r\n");
        }
        sbType.Append("\r\n");
    }

    // Additional foreach loops for CSpace, SSpace, and CSSpace
    ...

    sbType.Append("\r\n");
    string Ctypes = sbType.ToString();
    // Save to a text file
    StreamWriter mdFile = File.CreateText("MetaDataContext.txt");
    mdFile.WriteLine(sbType.ToString());
    mdFile.Flush();
    mdFile.Close();
}
```

### VB 9.0

```
Private Sub SaveWorkspaceMetadata()
    Dim nwMdws As MetadataWorkspace = ctxNwind.MetadataWorkspace
    Dim nwTypes As ReadOnlyCollection(Of EdmType) = _
        nwMdws.GetItems(Of EdmType)(DataSpace.OSpace)
    sbType.Append("ObjectItemCollection (OSpace)" & vbCrLf)
```

## Part IV: Introducing the ADO.NET Entity Framework

---

```
For Each nwType As EdmType In nwTypes
    sbType.Append("ObjectItem: " & nwType.GetType().FullName & _
    ", " & nwType.FullName & vbCrLf)
    For Each nwProp As MetadataProperty In nwType.MetadataProperties
        sbType.Append("Property: " & nwProp.Name & _
        (If((nwProp.Value IsNot Nothing), ", " & _
        nwProp.Value.ToString(), "") & vbCrLf)
    Next nwProp
    sbType.Append(vbCrLf)
Next nwType
' ...
sbType.Append(vbCrLf)
Dim Ctypes As String = sbType.ToString()
Dim mdFile As StreamWriter = File.CreateText("MetaDataCollections.txt")
mdfile.WriteLine(sbType.ToString())
mdfile.Flush()
mdfile.Close()
End Sub
```

The sample files include the `MetaDataCollections.txt` file, which has about 5,500 lines for their EDMs and contain all eight original Northwind tables.

### ObjectStateManager

Each `ObjectContext` has its own `ObjectStateManager` instance, which performs the following functions:

- Manages the identity of entity instances in the `ObjectContext`'s data cache by referring to `EntityKey` values
- Assures that only one copy of a particular entity is present in the data cache by testing `EntityKey` values
- Provides change tracking by maintaining the current state of each entity instance in the data cache: `Added`, `Deleted`, `Detached`, `Modified`, or `Unchanged`
- Provides a change set to manage concurrency conflicts for updates to entity instances

The `ObjectStateManager` ordinarily is for internal use only. However, you can use its `GetObjectStateEntries()` method to return a collection of `ObjectStateEntry` objects for a specified `EntityState`, or the `GetObjectStateEntry()` or `TryGetObjectStateEntry()` method to return the corresponding `ObjectStateEntry` for a specified `EntityKey`.

### Writing ObjectQueries with Entity SQL

Entity SQL statements for `ObjectQuery<EntityType>` queries are identical to those for `EntityQuery`s that return untyped client views but execution syntax differs. As mentioned earlier, execution of `ObjectQuery<EntityType>` queries is deferred.

If the default `MergeOption.AppendOnly` property value, which appends new entries only to the `ObjectStateManager`, is satisfactory, you can use the following simplified form of a parameterized

## Chapter 9: Raising the Level of Data Abstraction

---

SELECT VALUE query and execute it by assigning as the `DataSource` property value of a `BindingSource`, `EntityDataSource`, `DataGridView`, or other data-aware control or component:

### C# 3.0

```
string esqlCusts = "SELECT VALUE Cust FROM NorthwindEntities.Customers AS Cust" +
    " WHERE Cust.Country = @Country";
ObjectQuery<Customer> nwCusts =
    ctxNwind.CreateQuery<Customer>(esqlCusts,
        new ObjectParameter("Country", "USA"));
customerBindingSource.DataSource = nwCusts;
```

### VB 9.0

```
Dim esqlCusts As String = "SELECT VALUE Cust " & _
    "FROM NorthwindEntities.Customers AS Cust" & _
    " WHERE Cust.Country = @Country"
Dim nwCusts As ObjectQuery(Of Customer) = _
    ctxNwind.CreateQuery(Of Customer)(esqlCusts,
        New ObjectParameter("Country", "USA"))
customerBindingSource.DataSource = nwCusts;
```

If you need a different `MergeOption`, such as `NoTracking` for high-performance, read-only applications, or `OverwriteChanges` or `PreserveChanges` for concurrency management, create a new `ObjectQuery(Of T)` and new `ObjectParameter`, as in these examples:

### C# 3.0

```
string esqlOrds = @"SELECT VALUE Ord FROM NorthwindEntities.Orders AS Ord" +
    " WHERE Ord.Customer.Country = @Country " +
    "ORDER BY Ord.OrderID DESC";
ObjectQuery<Order> nwOrds = new ObjectQuery<Order>(esqlOrds, ctxNwind,
    MergeOption.NoTracking);
ObjectParameter prmCountry = new ObjectParameter("Country", "USA");
nwOrds.Parameters.Add(prmCountry);
orderBindingSource.DataSource = nwOrds;
```

### VB 9.0

```
Dim esqlOrds As String = "SELECT VALUE Ord " & _
    "FROM NorthwindEntities.Orders AS Ord" & _
    "WHERE Ord.Customer.Country = @Country ORDER BY Ord.OrderID DESC"
Dim nwOrds As ObjectQuery(Of Order) = _
    New ObjectQuery(Of Order)(esqlOrds, ctxNwind, MergeOption.AppendOnly)
Dim prmCountry As New ObjectParameter("Country", "USA")
nwOrds.Parameters.Add(prmCountry)
orderBindingSource.DataSource = nwOrds;
```

*Notice that the `ToList()` method isn't invoked when assigning the `ObjectQuery` to the `BindingSource` controls in this and the succeeding examples. If you append the `ToList()` method object additions and deletions don't persist to the data store when you invoke the `SaveChanges()` method. The `ObjectStateManager` doesn't track new or deleted `List<T>` items.*

Figure 9-10 shows the `NwindObjectServicesCS.sln` sample project's UI. By default, the project executes Entity SQL `ObjectQuery<Customer>` and `ObjectQuery<Order>` queries from the `FormMain_Load`

## Part IV: Introducing the ADO.NET Entity Framework

event handler with lazy loading of related Order entities in the customerBindingSource\_CurrentChanged event handler. You can choose to eager load all customers and orders by marking the Cache Customers' Orders check box. Marking the Use Query Builder check box substitutes Query builder methods for Entity SQL; marking the Use LINQ Queries check box substitutes LINQ to Entities for query builder methods or Entity SQL.

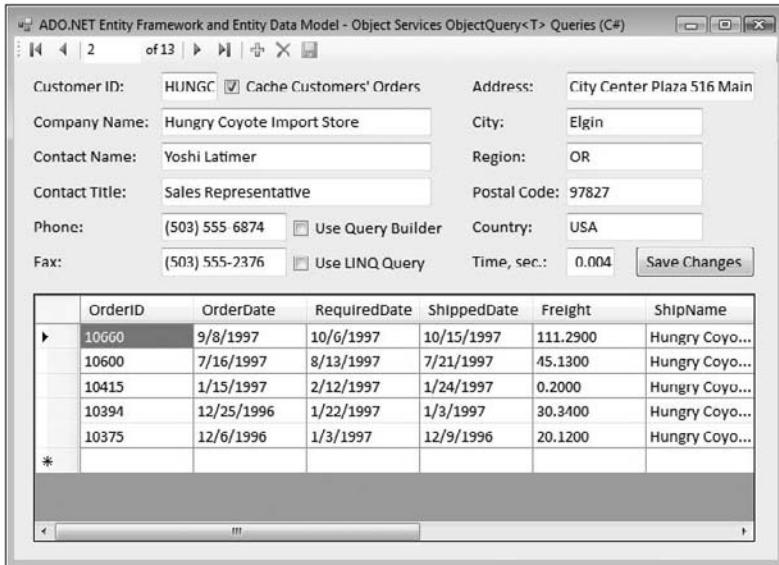


Figure 9-10

*Check-box selections are persisted in the App.config file's settings group.*

Following are other available operations in the sample project's UI:

- ❑ Clicking the Add (+) navigator button and confirming the addition adds new default Customer and Order entities to the DataContext.
- ❑ Editing Customer and Order data in the text boxes and DataGridView updates the DataContext.
- ❑ Clicking the Delete (x) navigator button deletes the Customer and associated Order entities from the DataContext. (The ObjectStateManager performs a cascade deletion.)
- ❑ Adding a new Order in the DataGridView adds it to the DataContext.
- ❑ Selecting an Order in the DataGridView and pressing Delete deletes it from the DataContext.
- ❑ Clicking Save Data updates the data store and regenerates the ObjectContext.

Chapter 14 describes displaying and updating entities with data-aware Windows and Web controls and components, including the ASP.NET EntityDataSource.

### Composing ObjectQueries with Query Builder Methods

Query builder methods (QBM) are a composable set of functions with Pascal-cased Entity SQL names that you can chain to create the equivalent of complete Entity SQL statements. Each QBM returns a new ObjectQuery instance. Following are the QBM and their corresponding Entity SQL statements:

Query Builder Method	Entity SQL Statement	Query Builder Method	Entity SQL Statement
Distinct	DISTINCT	SelectValue	SELECT VALUE
Except	EXCEPT	Skip	SKIP
GroupBy	GROUP BY	Top	TOP and LIMIT
Intersect	INTERSECT	Union	UNION
OfType	OFTYPE	UnionAll	UNION ALL
OrderBy	ORDER BY	Where	WHERE
Select	SELECT		

You can terminate QBM chains with LINQ Standard Query Operators (SQOs), such as `First`, `FirstOrDefault`, `Count`, and aggregate operators. A LINQ SQO can't precede a QBM, because LINQ doesn't return the `ObjectQuery` type.

*The primary benefit of QBM queries is immunity to SQL injection attacks. The performance difference between QBM, conventional Entity SQL, and LINQ to Entities queries isn't substantial.*

The sample project's `MainForm_Load` event handler includes the following code to load a subset of the `Customers` `EntitySet` into the `customerBindingSource` and related `Order` instances with a descending `OrderID` sort into the data cache:

#### C# 3.0

```
ObjectQuery<Customer> nwCusts =
    ctxNwind.Customers.Where("it.Country = @Country",
        new ObjectParameter("Country", "USA"));
customerBindingSource.DataSource = nwCusts;

// QueryByDescending method is missing; LINQ descending sort required
ObjectQuery<Order> nwOrds =
    ctxNwind.Orders.Where("it.Customer.Country = @Country",
        new ObjectParameter("Country", "USA"))
        .OrderBy("it.OrderID DESC");
orderBindingSource.DataSource = nwOrds;
```

## Part IV: Introducing the ADO.NET Entity Framework

---

### VB 9.0

```
Dim nwCusts As ObjectQuery(Of Customer) = _
    ctxNwind.Customers.Where("it.Country = @Country", _
        New ObjectParameter("Country", "USA"))
customerBindingSource.DataSource = nwCusts

' QueryByDescending method is missing; LINQ descending sort required
Dim nwOrds As ObjectQuery(Of Order) = _
    ctxNwind.Orders.Where("it.Customer.Country = @Country", _
        New ObjectParameter("Country", "USA"))
        .OrderBy("it.OrderID DESC")
orderBindingSource.DataSource = nwOrds
```

*Query builder method syntax is quite similar to LINQ to Entities method call syntax but doesn't use lambda functions. The it pronoun that precedes a property name refers to the current instance of the ObjectQuery's EntitySet, so it corresponds to x => x or Function(x) c.*

If you don't need the descending sort, you can simplify the preceding code by invoking the `Include ("Orders")` method on the parent entity to specify a query path and cache the related Order instances:

### C# 3.0

```
ObjectQuery<Customer> nwCusts =
    ctxNwind.Customers.Include("Orders")
        .Where("it.Country = @Country",
            new ObjectParameter("Country", "USA"));
customerBindingSource.DataSource = nwCusts;
```

### VB 9.0

```
Dim nwCusts As ObjectQuery(Of Customer) = _
    ctxNwind.Customers.Include("Orders").Where("it.Country = @Country", _
        New ObjectParameter("Country", "USA"))
customerBindingSource.DataSource = nwCusts
```

The `Include()` method, which is restricted to use with query builder method and LINQ to Entities queries, bears a resemblance to LINQ to SQL's `DataContext.LoadOptions` properties and `DataContext.AssociateWith()` method. However, LINQ to SQL's `AssociateWith()` method lets you customize the associated entity set with LINQ `Where`, `OrderBy`, `ThenBy`, `OrderByDescending`, `ThenByDescending`, and `Take` SQOs; `Include()` doesn't.

## Using the LINQ to Entities Provider

LINQ to Entities generates an expression tree and transforms it to a CQT that passes to the store-specific ADO.NET data provider for execution by the RDBMS. LINQ to Entities's method call syntax delivers the tersest of all `ObjectQuery` expressions.

*Contrary to a few blog posts and MSDN ADO.NET (Pre-release) forum questions, EF doesn't translate LINQ to Entities into Entity SQL for translation to CQT. The translation process would impair performance significantly.*

The LINQ to Entities implementation supports all but a few LINQ SQOs and their overloads. For example, `Aggregate`, `Contains`, `DefaultIfEmpty`, `ElementAt`, `ElementAtOrDefault`, `Last`, `LastOrDefault`, `Reverse`, `Single`, `SingleOrDefault`, `SkipWhile`, and `TakeWhile` SQOs aren't supported. LINQ to Entities doesn't support overloads that enable choosing a custom equality comparer with the `IEqualityComparer<TSource>` interface or an integer argument that represents the index of the sequence on which the operation is conducted.

LINQ to Entities queries are identical in most cases to LINQ to SQL queries for the same result sequence. Following are the LINQ to Entities expressions to eager-load `Order` instances for the U.S. `Customers` subset:

### C# 3.0

```
var nwCusts = from c in ctxNwind.Customers
              where c.Country == "USA"
              select c;
customerBindingSource.DataSource = nwCusts;

var nwOrds = from o in ctxNwind.Orders
              where o.Customer.Country == "USA"
              orderby o.OrderID descending
              select o;
orderBindingSource.DataSource = nwOrds;
```

### VB 9.0

```
Dim nwCusts = From c In ctxNwind.Customers _
               Where c.Country = "USA" _
               Select c
customerBindingSource.DataSource = nwCusts

Dim nwOrds = From o In ctxNwind.Orders _
              Where o.Customer.Country = "USA" _
              Order By o.OrderID Descending _
              Select o
orderBindingSource.DataSource = nwOrds
```

Here are the corresponding method call syntax queries:

### C# 3.0

```
var nwCusts = ctxNwind.Customers.Where(c => c.Country == "USA");
customerBindingSource.DataSource = nwCusts;

var nwOrds = ctxNwind.Orders.Where(o => o.Customer.Country == "USA")
                  .OrderByDescending(o => o.OrderID);
orderBindingSource.DataSource = nwOrds;
```

### VB 9.0

```
Dim nwCusts = ctxNwind.Customers.Where(Function(c) c.Country = "USA")
customerBindingSource.DataSource = nwCusts

Dim nwOrds = ctxNwind.Orders.Where(Function(o) o.Customer.Country = "USA") _
                  .OrderByDescending(Function(o) o.OrderID)
orderBindingSource.DataSource = nwOrds
```

## Part IV: Introducing the ADO.NET Entity Framework

---

*You can compile LINQ to Entities queries to maximize performance of successive executions.*

*The process is similar to that for compiling LINQ to SQL queries.*

Chapter 12 provides more detailed information about LINQ to Entities queries against complex ObjectContexts that have polymorphic entities with the three types of inheritance. Chapter 12 also shows you how to compile LINQ queries and provides a few examples of execution time savings.

## Understanding the Persistence Ignorance Controversy

Swedish developer and author Jimmy Nilsson addressed the issue of keeping infrastructure-related code, such as attributes required by O/RM for persistence operations, out of domain objects in the “POCO as a Lifestyle” topic of his best-selling *Applying Domain Driven Design and Patterns With Examples in C# and .NET* book. Martin Fowler, author of *Patterns of Enterprise Application Architecture* and several other widely read treatises on OO architecture, suggested the term *Persistence Ignorance* (PI) as being clearer than Plain Old CLR Objects (POCO) or the Java equivalent, Plain Old Java Objects (POJO.)

*EF implements Fowler's Data Mapper, Unit of Work, Identity Mapper, QueryObject, Foreign Key Mapping and Lazy Load patterns. Fowler defines the Data Mapper pattern as “a layer of software that separates the in-memory objects from the database. . . . With Data Mapper, the in-memory objects needn't know even that there's a database present.” You can learn more about these patterns at [www.martinfowler.com/eaaCatalog](http://www.martinfowler.com/eaaCatalog).*

Nilsson listed in Chapter 7 the following eight restrictions (paraphrased) that a persistent-ignorant framework *must not impose* on the developer's domain objects:

1. Require inheriting from a specified base class other than `Object`
2. Require use of a factory method to instantiate them
3. Require use of specialized data types for properties, including collections
4. Require implementing particular interfaces
5. Require use of framework-specific constructors
6. Require use of specific fields or properties
7. Prohibit specific application constructs
8. Require writing RDBMS code, such as queries or stored procedure calls in a provider-specific dialect

The attitudes of .NET architects and developers run the gamut from *harboring the belief that applications should be able to substitute relational or object databases, Web services, Plain Old XML (POX) files, or even comma-separated text files for domain object persistence without changing a single line of source code to having no interest whatsoever in PI*. PI simplifies test-driven development (TDD) and mocking of O/RM tools and RDBMSs, so there's an increasing proportion of developers who favor as much PI as they can obtain without impacting performance dramatically or writing their own O/RM framework.

NHibernate provides total PI by not requiring the domain object layer to require a reference to `NHibernate.dll`. On the other hand, the general consensus of .NET developers appears to be that NHibernate is very complex and requires a great deal of effort to craft entity code together with mapping files, attributes or both.

Some data-centric application patterns don't achieve PI intentionally. For example, a popular pattern, which Fowler calls *Active Record*, requires a one-to-one relationship between domain objects and tables and puts the table access logic into the domain model. Therefore, the Active Record pattern dictates intimacy with the data store. Similarly the LLBLGen Pro O/RM framework ignores PI principles by implementing an `IConcurrencyPredicateType` interface in domain entities.

EF's architects didn't consider PI in the initial design. (The same is true for LINQ to SQL; however, LINQ to SQL enables POCO.) About midway in the development of EF v1, the ADO.NET team got the PI message from a group of Microsoft Most Valued Professionals (MVPs) that became known as the "NHibernate Mafia." EF v1 includes the first steps toward the goal of full PI as an option in v2. These first steps were to enable implementation of IPOCO (Interface POCO) by creating custom classes that inherit from the following interfaces:

- `IEntityWithChangeTracker` for change tracking
- `IEntityWithKey` for exposing the entity identity key (optional, but not implementing this interface causes a significant reduction in performance)
- `IEntityWithRelationships` required for entities with associations

Of course, this approach trades inheriting from the specified `EntityObject` base class for implementing particular interfaces, but it's a step in the right direction. Custom classes and their properties must have the appropriate EDM attributes applied. You must write the code for your custom classes; you can't depend on `EdmGen.exe` or the EDM Wizard to generate it for you. On the other hand, many developers consider code generated by O/R Mapping tools to be suspect as prolix and hard to maintain. If you decide you must have 100 percent PI and oppose code generation on principle, EF isn't the O/RM tool for you.

If you're willing to sacrifice PI in v1 and accept code generation for rapid application development, EF should be a top contender as your O/RM solution for medium to large-scale .NET projects. Otherwise, you can wait for EF v2, which is expected to be PI-compliant.

## Summary

The ADO.NET Entity Framework is the founding member of a putative Microsoft Entity Data Platform. Microsoft representatives have said little or nothing more about the Entity Data Platform since the October 2007 Microsoft Business Intelligence Conference. However, Microsoft is making a substantial investment in architecting, developing, and evangelizing v1 of its second attempt to create a versatile, enterprise-grade O/RM and persistence framework.

## Part IV: Introducing the ADO.NET Entity Framework

---

Microsoft insists that EF is more than an O/RM tool because of its emphasis on the Entity Data Model. Even if Microsoft doesn't "productize" the Entity Data Platform, it's likely that other data-related products, such as Reporting Services and Synchronization Services, will adopt the EDM and Entity SQL with or without Object Services and LINQ to Entities. EDM with Object Services will enable any RDBMS with a provider-specific ADO.NET data provider to take advantage of EF's strongly typed entities.

The chapter covered the basic details of Dr. Peter Chen's entity/relationship model as it relates to EF and then went on to describe ADNEF's primary features: the EDM Designer, mapping schemas, EntityClient, Entity SQL, Client Views, Object Services, ObjectQueries, the query builder method and LINQ to Entities queries. C#, and VB sample projects demonstrate working implementations of these features, as well as databinding with BindingSource components to text boxes and DataGridView controls.

The remaining chapters and sample code of Part 4 and all of Part 5 expand greatly on the topics introduced in this chapter.

# 10

## Defining Storage, Conceptual, and Mapping Layers

The primary feature that distinguishes the ADO.NET Entity Framework (EF) from the myriad other object/relational mapping (O/RM) tools for .NET is its layered approach to the mapping process for creating an Entity Data Model (EDM). Layered mapping prevents simple changes to the relational data store's schema from causing changes in the EDM object graph's structure that force source code modifications. For example, DBAs can split or combine tables without requiring rewriting code if the changes can be accommodated by modifying the XML content of the storage and mapping layers only. Entity SQL (eSQL) queries at the `EntityClient` level and eSQL or LINQ to Entities queries at the `ObjectServices` level use the same mapping files.

The EDM is part of Microsoft's wide-ranging Entity Data Platform strategy that Michael Pizzo, a data architect in Microsoft's Data Programmability Group, described in an April 2007 blog post entitled "Microsoft's Data Access Strategy."

Microsoft envisions an Entity Data Platform that enables customers to define a common Entity Data Model across data services and applications. The Entity Data Platform is a multi-release vision, with future versions of reporting tools, replication, data definition, security, etc., all being built around a common Entity Data Model.

Successive releases are expected to include SharePoint lists, Web services, and well-established XML document formats, such as Atom, InfoPath, or Open XML (OXML) as data sources rather than persistence stores.

*It's reasonable to expect similarity among data sources in the EDM Designer surface, the top-level conceptual (CSDL) layer's XML content, and the autogenerated classes for `EntityType`s. However, there undoubtedly will be wide variation in the schemas for the mapping (MSL) and*

## Part IV: Introducing the ADO.NET Entity Framework

---

*physical (SSDL) layers. Thus total persistence ignorance for EF, which is the focus of object/relational purists, isn't likely to be found with nonrelational data sources.*

Chapter 9 introduced the three XML files that map the conceptual EDM to the data store's relational model. The EDM Designer or `EdmGen.exe` produces adequate mapping files for the initial version of most projects and their entities, but you'll find that you must manually edit the `ModelName.edmx` file to take one or more of the following activities:

- ❑ Defining read-only entities with `<QueryView>` groups
- ❑ Fixing errors generated by removing in the EDM Designer extraneous composite primary key values inferred for entities created from views
- ❑ Updating mapping of custom `Entity.Property` names that don't match `Table.Column` names
- ❑ Implementing *complex types* (also called *value types*)
- ❑ Making edits that prevent you from opening `ModelName.edmx` in the EDM Designer

Some changes you make to entity properties in the designer or directly to the underlying `ModelName.edmx` file might render it unusable with the designer, in which case a marquee suggests that you open the file in VS 2008's XML Editor. Alternatively, you might need to edit a `ModelName.ssdl`, `ModelName.msdl`, or `ModelName.sSDL` runtime file, or — even worse — write all three files from scratch. The first part of this chapter delivers the information about EF's XML mapping files, which you need to take full advantage of EF.

*Back up the `ModelName.edmx` file early and often as you make changes to the EDM in the designer. If your alterations cause the file to become incompatible with the designer, you might lose the ability to undo your changes. If you don't have a backup, you must edit the `ModelName.edmx` file in VS's XML Editor to repair the damage, which might be time-consuming.*

EF's `MetadataWorkspace` class holds the runtime representation of the metadata from the three mapping files and autogenerated partial classes from the `ModelName.Designer.cs` or `.vb` file. Obtaining the names and properties of many EDM components, which Microsoft calls `Items`, requires accessing — and often iterating — the items of the appropriate `EntityContainer`'s `DataSpace`. For example, if you attach objects and want to mark their properties as modified, you must apply the `SetPropertyModified()` method to every `EdmProperty` for the specified `EntityType` in the `MetadataWorkspace`. Therefore, this chapter also shows you how to access and, where possible, modify properties of `MetadataWorkspace` `Items`.

Figure 10-1 shows the relationships between the objects and resources covered by this chapter.

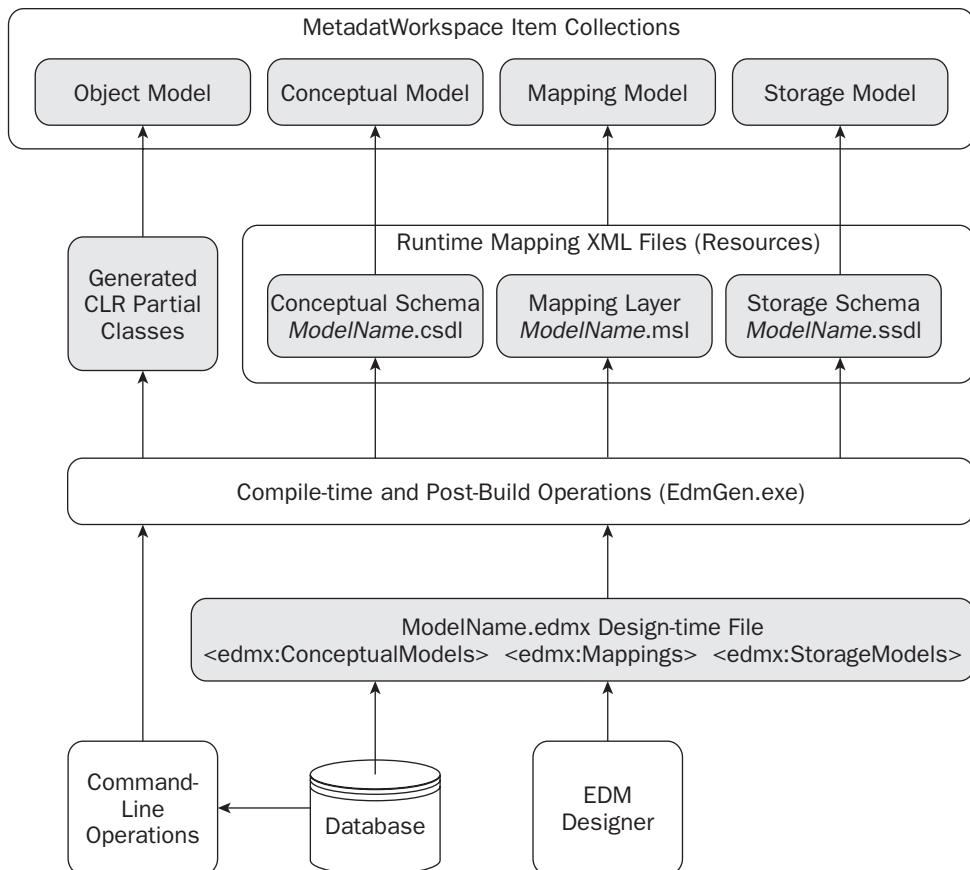


Figure 10-1

## Exploring and Customizing the EDMX File

The XML Editor has references to XML schemas for the EDMX and runtime files; the schemas enable IntelliSense. To display the design-time *ModelName.edmx* EDM Designer file in the XML Editor, right-click the file in Solution Explorer, choose Open With to open the eponymous dialog, and double-click XML Editor to display the file. Choose XML, Schemas to open the XML Schemas dialog; schemas marked with a green check in the Use column match the *ModelName.edmx*'s `xmlns:prefix` attribute values (see Figure 10-2). Scroll down to the `urn:schemas-microsoft-com:windows:storage:mapping:CS` schema, which isn't visible in Figure 10-2.

## Part IV: Introducing the ADO.NET Entity Framework

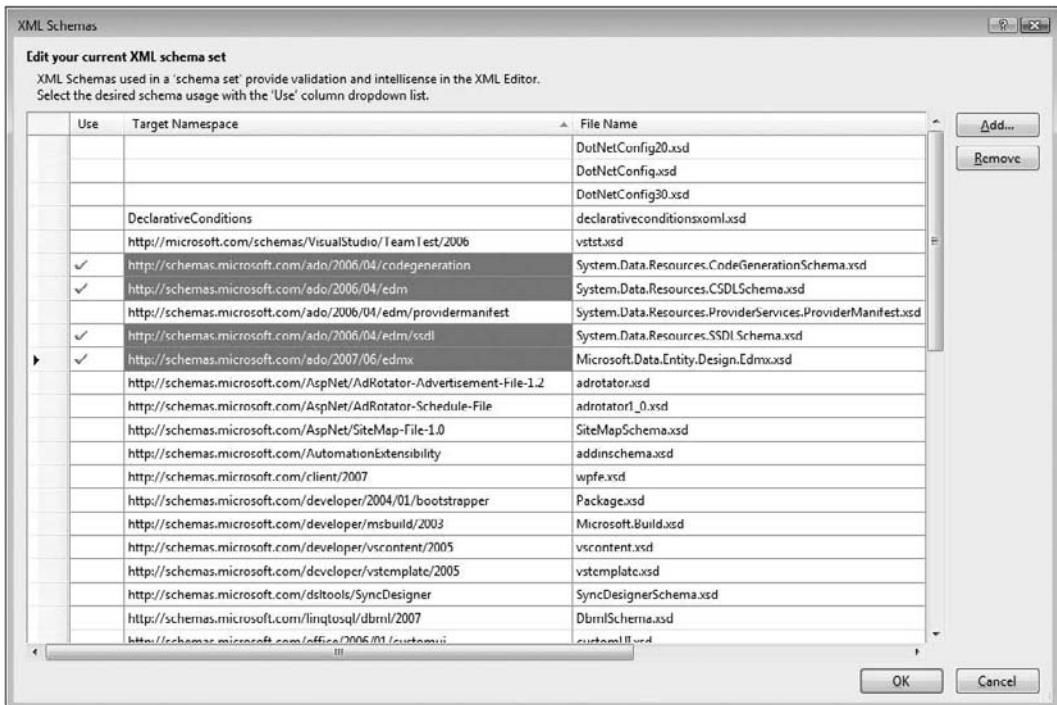


Figure 10-2

*IntelliSense applies only to elements or attributes you add to the schema, not to edits you make to existing elements or attributes.*

The `ModelName.edmx` file validates to the `http://schemas.microsoft.com/ado/2007/06/edmx` schema and consists of a hierarchy of the following XML sections:

- ❑ The `<edmx:Designer>` section contains groups and elements to specify designer-specific property values. These properties aren't exposed as .NET objects at runtime.
- ❑ The `<edmx:Runtime>` section contains the following three subsections:
  - ❑ The `<edmx:ConceptualModels>` section is the source of the runtime `modelName.csdl` file and `MetadataWorkspace`'s `CSpace` Items and validates to the `http://schemas.microsoft.com/ado/2006/04/edm` schema.
  - ❑ The `<edmx:StorageModels>` section is the source of the runtime `modelName.ssdl` file and `MetadataWorkspace`'s `SSpace` Items and validates to the `http://schemas.microsoft.com/ado/2006/04/edm/ssdl` schema.
  - ❑ The `<edmx:Mapping>` section is the source of the runtime `modelName.msl` file and `MetadataWorkspace`'s `CSSpace` Items and validates to the `urn:schemas-microsoft-com:windows:storage:mapping:CS` schema.

## Chapter 10: Defining Storage, Conceptual, and Mapping Layers

---

*The XML Schema files for the three preceding urns are System.Data.Resources.CSDLSchema.xsd, System.Data.Resources.SSDLSchema.xsd, and System.Data.Resources.CSMSL.xsd in the \Program Files\Microsoft Visual Studio 9.0\XML\Schema folder. You also can view them with Red Gate's (formerly Lutz Roeder's) Reflector application in the System.Data.Entity namespace's Resources section.*

A post-build operation copies the content of the three <edmx:Runtime> sections to the executable folder (... \bin\debug for Debug mode) of Windows Form and Web Application projects. The operation overwrites previous versions without a warning message. File-system Web Site projects store the three files in the project's DLL as resources.

*The MetadataWorkspace also has OSpace Items for the autogenerated object model and OCSpace Items for mapping between the conceptual and object models. These DataSpaces are one of the subjects of the "Traversing the MetadataWorkspace" section near the end of the chapter.*

The following sections describe the EDMX file's content for the NwindProductsSProcsCS.sln sample EDM application based on the Northwind sample database's Products, Categories and Suppliers tables together with optional stored procedures for data retrieval and update (CRUD) operations. You'll find CS and VB project versions of this project in the \WROX\ADONET\Chapter13\CS and ...VB folders. The T-SQL script to create the stored procedures in the Northwind sample database is \WROX\Chapter10\NwindProductsStoredProcs.sql. Sample projects' EntityType names are singularized, EntitySet names are pluralized, and navigation Role names are adjusted to reflect the association's multiplicity, as noted in Chapter 9.

*The first section is "Storage Models," rather than "Conceptual Models," because EF v1 generates EDMs from databases only (also called bottom-up generation). An early goal during the feature's development was to generate the underlying database from the conceptual model (top-down). Like several other features, such as support for SQL Server's inline table-valued functions (TVFs), this feature was postponed to EF v2 because of a lack of resources, to meet the v1 ship schedule, or both.*

## Storage Models (SSDL Content)

You determine the *ModelName.ssdl* content with the EDM Wizard when you select the tables, views, scalar-value functions, and stored procedures to include in the EDM. Updating, adding or deleting database objects with the EDM Wizard, or including the use of the Update Model from Database feature, also modifies the SSDL section.

*To synchronize the EDM with changes to the database schema, right-click the Model Browser and choose Update Model from Database to open a modified version of the Wizard's Choose Your Database Objects dialog. This dialog contains Add, Update, and Delete nodes for the three database object types, as described in Chapter 9's "Adding, Updating, and Deleting the Model's Objects" topic.*

*Synchronizing an EDMX file that contains custom edits to the <edmx:StorageModels> section might prevent the file from validating and require redoing previous edits.*

## Part IV: Introducing the ADO.NET Entity Framework

---

### The EntityContainer Subgroup

The `<Schema>` group's `<EntityContainer>` subgroup contains the following group hierarchy:

```
<EntityContainer Name="DatabaseNameEntities">
  <EntitySet Name="EntitySetName" EntityType="ModelName.Store.TableName" />
  ...
  <EntitySet Name="EntitySetName#Name" EntityType="ModelName.Store.Table#Name" >
    <DefiningQuery>
      SELECT ... FROM ... <!--SELECT statement for view as source -->
    </DefiningQuery>
  </EntitySet>
  <AssociationSet Name="ForeignKeyName#" Association="ModelName.Store.ForeignKeyName#">
    <End Role="Role1Name" EntitySet="EntitySetName" />
    <End Role="Role2Name" EntitySet="EntitySetName" />
  </AssociationSet>
  ...
</EntityContainer>
```

Databases with multiple schemas generate an `<EntityContainer>` subgroup for each database schema. For example, adding all objects from the AdventureWorks sample database creates five `<EntityContainer>` groups, one for each schema (Human Resources, Person, Production, Purchasing, and Sales) in a 7,801-line `AdventureWorks.edmx` file. (The autogenerated `AdventureWorks.Designer.cs` file has 24,671 lines.)

### EntitySet Elements or Subgroups

An `<EntitySet>` element or subgroup must be present for each database table or view, regardless of the hierarchical model for the table(s) chosen at the conceptual level, if any. The `Store` component of the `<EntityType>` name identifies the `<EntitySet>` as a member of the `MetadataWorkspace`'s `SSpace` `DataSpace`.

`<DefiningQuery>` elements are present only for views and consist of a simplified version of the `SELECT` statement that includes the column-list and table-name only; `TOP`, `WHERE`, `ORDER BY`, and other modifiers are missing.

*If the domain model implements entity inheritance by deriving entities from a base type, `<EntitySet>` elements for the base type table and additional tables to support derived types for table-per-type (TPT) or table-per-concrete-type (TCT) inheritance, or multiple-entity-sets-per-type (MEST) structures for partitioning would appear in the `<EntityContainer>` subgroup. This chapter deals only with tables that have a 1:1 relationship with types or table-per-hierarchy (TPH) types in which properties for all derived types are stored in a single table with a discriminator column. Chapter 13 shows you how to implement TPT, TCT, and MEST inheritance models.*

### AssociationSet Elements or Subgroups

`<AssociationSet>` subgroups define association pairs named for the relationship; binary associations involve two tables in the storage model. It's a common practice to remove the `FK_` prefix from SQL Server's autogenerated foreign-key constraint name because entity-relationship nomenclature doesn't include foreign keys. When you change the `Association Set Name` and `(Association) Name` property values in the Properties sheet, only the conceptual layer names change.

# Chapter 10: Defining Storage, Conceptual, and Mapping Layers

---

Like the <EntitySet>, the `Store` component of the <Association> name identifies the <EntitySet> as a member of the `MetadataWorkspace`'s `SSpace` `DataSpace`; the <Association> attribute would be better named <AssociationType> for consistency.

Mapping is based on the `Role` element's `Name` attribute value.

## The EntityContainer Subgroup for Categories, Products, and Suppliers

Following is the content of the EntityContainer subgroup for the sample project's `Products.edmx` file:

```
<Schema Namespace="NorthwindModel.Store" Alias="Self"
    ProviderManifestToken="09.00.3054"
    xmlns="http://schemas.microsoft.com/ado/2006/04/edm/ssdl">
    <EntityContainer Name="dbo">
        <EntitySet Name="Categories"
            EntityType="NorthwindModel.Store.Categories" />
        <EntitySet Name="Products" EntityType="NorthwindModel.Store.Products" />
        <EntitySet Name="Suppliers" EntityType="NorthwindModel.Store.Suppliers" />
        <EntitySet Name="uvw_GetAllProducts"
            EntityType="NorthwindModel.Store.uvw_GetAllProducts">
            <DefiningQuery>
                SELECT
                    [uvw_GetAllProducts].[ProductID] AS [ProductID],
                    [uvw_GetAllProducts].[ProductName] AS [ProductName],
                    [uvw_GetAllProducts].[SupplierID] AS [SupplierID],
                    [uvw_GetAllProducts].[CategoryID] AS [CategoryID],
                    [uvw_GetAllProducts].[QuantityPerUnit] AS [QuantityPerUnit],
                    [uvw_GetAllProducts].[UnitPrice] AS [UnitPrice],
                    [uvw_GetAllProducts].[UnitsInStock] AS [UnitsInStock],
                    [uvw_GetAllProducts].[UnitsOnOrder] AS [UnitsOnOrder],
                    [uvw_GetAllProducts].[ReorderLevel] AS [ReorderLevel],
                    [uvw_GetAllProducts].[Discontinued] AS [Discontinued]
                FROM [dbo].[uvw_GetAllProducts] AS [uvw_GetAllProducts]
            <!--Errors Found During Generation: warning 6002:
                The table/view 'Northwind.dbo.uvw_GetAllProducts' does not have a
                primary key defined. The key has been inferred and the definition
                was created as a read-only table/view. -->
            </DefiningQuery>
        </EntitySet>
        <AssociationSet Name="FK_Products_Categories"
            Association="NorthwindModel.Store.FK_Products_Categories">
            <End Role="Categories" EntitySet="Categories" />
            <End Role="Products" EntitySet="Products" />
        </AssociationSet>
        <AssociationSet Name="FK_Products_Suppliers"
            Association="NorthwindModel.Store.FK_Products_Suppliers">
            <End Role="Suppliers" EntitySet="Suppliers" />
            <End Role="Products" EntitySet="Products" />
        </AssociationSet>
    </EntityContainer>
</Schema>
```

## Part IV: Introducing the ADO.NET Entity Framework

---

### **EntityType Subgroups**

Each `<EntitySet>` group has a corresponding `<EntityType>` subgroup of the `<Schema>` group. The `<EntityType>` subgroup represents the schema for the table that persists the corresponding entity. The data provider supplies the data type names and relates them to .NET types at the conceptual level. Following is the `<EntityType>` subgroup's hierarchy using generic data type names:

```
<Schema Namespace="ModelName.Store" Alias="Self"
    ProviderManifestToken="09.00.#####"
    xmlns="http://schemas.microsoft.com/ado/2006/04/edm/ssdl">
    ...
    <EntityType Name="SingularizedEntitySetName">
        <Key>
            <PropertyRef Name="PrimaryKeyFieldName" />
            ...
        </Key>
        <Property Name="AutogeneratedPrimaryKey" Type="NumberOrGuidType"
            Nullable="false"
            <StoreGeneratedPattern="Identity" />
        <Property Name="RequiredString" Type="characterType" Nullable="false"
            MaxLength="15" />
        <Property Name="OptionalString" Type="characterType" />
        ...
    </EntityType>
    ...
</Schema>
```

As noted in Chapter 9, it's a common practice to name `EntityType`s the singular value of the `EntitySet` name, and vice versa.

### **Key Subgroup**

Each member of an `EntitySet` must have a `Key` value for identity tracking. The `<Key>` subgroup's `<PropertyRef>` element specifies the primary key field `Name` value. The `EntitySet` accommodates composite primary keys. By default, the EDM Wizard infers composite primary keys for `EntityType`s based on views; inferred composite keys consist of the concatenated values of required fields.

*You can remove extraneous properties from the inferred composite key in the ModelName.edmx file's `<edmx:ConceptualModels>` section by right-clicking the property in the designer and choosing Entity Key to toggle the EntityKey property. However, doing this doesn't alter corresponding entries in the `<edmx:StorageModels>` section's `<Key>` group for the view. (The `<edmx:Mappings>` section doesn't include a `<Key>` group). If you receive an error message that includes text similar to "All the key fields (ViewName.PrimaryKeyId) of entity set ViewName must be mapped to all the key fields (ViewName.AllCompositeKeyFields) of table ViewName" as a result of this mismatch, manually remove or comment out the extraneous elements from the `<edmx:StorageModels>` section.*

### **Property Subelement**

`<Property>` elements correspond to fields of the underlying tables and share the field name by default. `Name` and `Type` attribute/scalar value pairs are required, all other attribute/value pairs are called *facets*, which apply to specific data types supported by the chosen RDBMS. The following table lists the `Name`, `description`, allowable values, default value and allowable T-SQL data types for SQL Server 2005+ facets.

## Chapter 10: Defining Storage, Conceptual, and Mapping Layers

---

Facet Name	Description	Allowable Values	Default Value	Allowable Data Types
Collation	String value indicating sorting and comparison of string values	Collations supported by data source	None	char, nchar, varchar, nvarchar
DateTimeKind	Enumeration of datetime offsets	UTC, Local, Unspecified	Unspecified	datetime
Default	Default value	Depends on data type	None	All except image
FixedLength	Length of fixed-width string	2047 max for Unicode, 4095 otherwise	None	
MaxLength	Maximum length variable-width string	2047 max for Unicode, 4095 otherwise	None	varchar, nvarchar
Nullable	Value required or not	true/false	true	All
Precision	Maximum number of digits of number	<= 38 for decimal, 19 for money (fixed)	18 for decimal, 19 for money	decimal, money
Preserve Seconds	Include seconds in datetime values	true/false	true	datetime
Scale	Maximum number of digits of decimal fraction	<= Precision for decimal, fixed for money	0 for decimal, 4 for money	decimal, money
StoreGenerated Pattern	Type of autogenerated value	None, Identity, Computed	None	int, bigint, unique identifier
Unicode	Unicode value or not	true/false	true	char, nchar, varchar, nvarchar

## Part IV: Introducing the ADO.NET Entity Framework

---

### The Product EntityType Subgroup

Following is the content of the <EntityType> subgroup for the Product type for SQL Server as the persistence store:

```
<Schema Namespace="NorthwindModel.Store" Alias="Self"
  ProviderManifestToken="09.00.3054"
  xmlns="http://schemas.microsoft.com/ado/2006/04/edm/ssdl">
  ...
  <EntityType Name="Product">
    <Key>
      <PropertyRef Name="ProductID" />
    </Key>
    <Property Name="ProductID" Type="int" Nullable="false"
      StoreGeneratedPattern="Identity" />
    <Property Name="ProductName" Type="nvarchar" Nullable="false"
      MaxLength="40" />
    <Property Name="SupplierID" Type="int" />
    <Property Name="CategoryID" Type="int" />
    <Property Name="QuantityPerUnit" Type="nvarchar" MaxLength="20" />
    <Property Name="UnitPrice" Type="money" />
    <Property Name="UnitsInStock" Type="smallint" />
    <Property Name="UnitsOnOrder" Type="smallint" />
    <Property Name="ReorderLevel" Type="smallint" />
    <Property Name="Discontinued" Type="bit" Nullable="false" />
  </EntityType>
  ...
</Schema>
```

The preceding type was chosen because it contains a variety of SQL Server data types and facets.

### Function Subelements and Subgroups

Substituting stored procedures for Entity SQL to populate (hydrate) EntitySets and T-SQL to persist changes to Entity objects the underlying data store isn't as simple as the process described in Chapter 5 for LINQ to SQL. Migrating to stored procedures in the EF is one of the primary topics of Chapter 13, so this chapter covers only those operations with the EDM Designer that affect the content of the \*.edmx file.

<Function> subelements and subgroups define the database's stored procedures and scalar functions in the following hierarchy for a scalar function and stored procedures for CRUD operations:

```
<Schema Namespace="modelName.Store" Alias="Self"
  ProviderManifestToken="09.00.####"
  xmlns="http://schemas.microsoft.com/ado/2006/04/edm/ssdl">
  ...
  <Function Name="ScalarFunctionName" ReturnType="datatype" Aggregate="false"
    BuiltIn="false" NiladicFunction="false" IsComposable="true"
    ParameterTypeSemantics="AllowImplicitConversion" Schema="owner" />
  <Function Name="RetrieveSProcName" Aggregate="false" BuiltIn="false"
    NiladicFunction="false" IsComposable="false"
    ParameterTypeSemantics="AllowImplicitConversion" Schema="owner">
    <Parameter Name="OptionalParamName" Type="datatype" Mode="In" />
  </Function>
```

## Chapter 10: Defining Storage, Conceptual, and Mapping Layers

```
<Function Name="InsertSPProcName" Aggregate="false" BuiltIn="false"
    NiladicFunction="false" IsComposable="false"
    ParameterTypeSemantics="AllowImplicitConversion" Schema="owner">
    <Parameter Name="Param1Name" Type="datatype" Mode="In" />
    ...
    <Parameter Name="ParamNName" Type="datatype" Mode="In" />
</Function>
<Function Name="UpdateSPProcName" Aggregate="false" BuiltIn="false"
    NiladicFunction="false" IsComposable="false"
    ParameterTypeSemantics="AllowImplicitConversion" Schema="OwnerName">
    <Parameter Name="PrimaryKey" Type="datatype" Mode="In" />
    <Parameter Name="Param2" Type="datatype" Mode="In" />
    ...
    <Parameter Name="ParamN" Type="datatype" Mode="In" />
</Function>
<Function Name="DeleteSPProcName" Aggregate="false" BuiltIn="false"
    NiladicFunction="false" IsComposable="false"
    ParameterTypeSemantics="AllowImplicitConversion" Schema="owner">
    <Parameter Name="PrimaryKeyName" Type="datatype" Mode="In" />
</Function>
...
</Schema>
```

The following table provides the description, allowable values, and default value of attributes for the preceding `<Function>` elements.

Attribute Name	Description	Allowable Values	Default Value
IsComposable	Specifies whether the return value can be used in FROM clause of other Entity SQL queries; true for scalar functions, false for stored procedures	true or false	true
IsAggregate	Specifies whether the function is an aggregate function (SUM, AVG, MIN, MAX, and so on)	true or false	false
IsNiladic	Specifies whether parenthesis can be omitted from a function call without a parameter (such as to T-SQL's CURRENT_TIMESTAMP) rather than to NewID(), which requires parenthesis	true or false	false

(continued)

## Part IV: Introducing the ADO.NET Entity Framework

Attribute Name	Description	Allowable Values	Default Value
IsBuiltIn	Specifies whether the function is specific to the data provider (SUM, AVG, MIN, MAX, and so on)	true or false	false
ParameterTypeSemantics	Specifies type semantics for promotability, implicit conversion, and explicit conversion for the provider primitive types: AllowImplicitConversion indicates an implicit conversion between the given and formal argument types, AllowImplicitPromotion indicates type promotion between the given and formal argument types, and ExactMatchOnly indicates strict equivalence is required.	AllowImplicitConversion, AllowImplicitPromotion or ExactMatchOnly	Allow Implicit Conversion

### Function Definitions for a DateTime Scalar Function and Product Stored Procedures

Following are definitions for a scalar function that returns the date of the first Order entity and stored procedures for CRUD functions on the Orders table:

```
<Schema Namespace="NorthwindModel.Store" Alias="Self"
  ProviderManifestToken="09.00.3054"
  xmlns="http://schemas.microsoft.com/ado/2006/04/edm/ssdl">
  ...
  <Function Name="ufn_GetFirstOrderDate" ReturnType="datetime"
    Aggregate="false" BuiltIn="false" NiladicFunction="false" IsComposable="true"
    ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo" />
  <Function Name="usp_GetCategoryByCategoryID" Aggregate="false" BuiltIn="false"
    NiladicFunction="false" IsComposable="false"
    ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">
    <Parameter Name="CategoryID" Type="int" Mode="In" />
  </Function>
  <Function Name="usp_InsertCategory" Aggregate="false" BuiltIn="false"
    NiladicFunction="false" IsComposable="false"
    ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">
    <Parameter Name="CategoryName" Type="nvarchar" Mode="In" />
    <Parameter Name="Description" Type="ntext" Mode="In" />
  </Function>
  <Function Name="usp_UpdateCategory" Aggregate="false" BuiltIn="false"
    NiladicFunction="false" IsComposable="false"
    ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">
    <Parameter Name="CategoryID" Type="int" Mode="In" />
    <Parameter Name="CategoryName" Type="nvarchar" Mode="In" />
```

# Chapter 10: Defining Storage, Conceptual, and Mapping Layers

```
<Parameter Name="Description" Type="ntext" Mode="In" />
</Function>
<Function Name="usp_DeleteCategory" Aggregate="false" BuiltIn="false"
    NiladicFunction="false" IsComposable="false"
    ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo">
    <Parameter Name="CategoryID" Type="int" Mode="In" />
</Function>
</Schema>
```

## Function Imports to Assign Data Retrieval Stored Procedures to an EntitySet

It's a common practice to create `EntitySet` and `EntityType` definitions in the EDM Designer for all tables of the source database regardless of the ultimate method used to populate the `EntitySets` at runtime — eSQL queries, LINQ to Entities or stored procedures. If you use a stored procedure for data retrieval, you can use the EDM Designer's Function Import feature to assign the stored procedure's resultset to the appropriate `EntityType` and rename the function, if desired.

For example, to map the `usp_GetProductByProductID` stored procedure to the `Product` `EntityType`, right-click the stored procedure item under the Model Browser pane's and choose Create Function Import to open the New Function Import dialog. Type the Function Import Name, and then open the Return Type list and select the `EntityType`, `Product` for this example (see Figure 10-3).

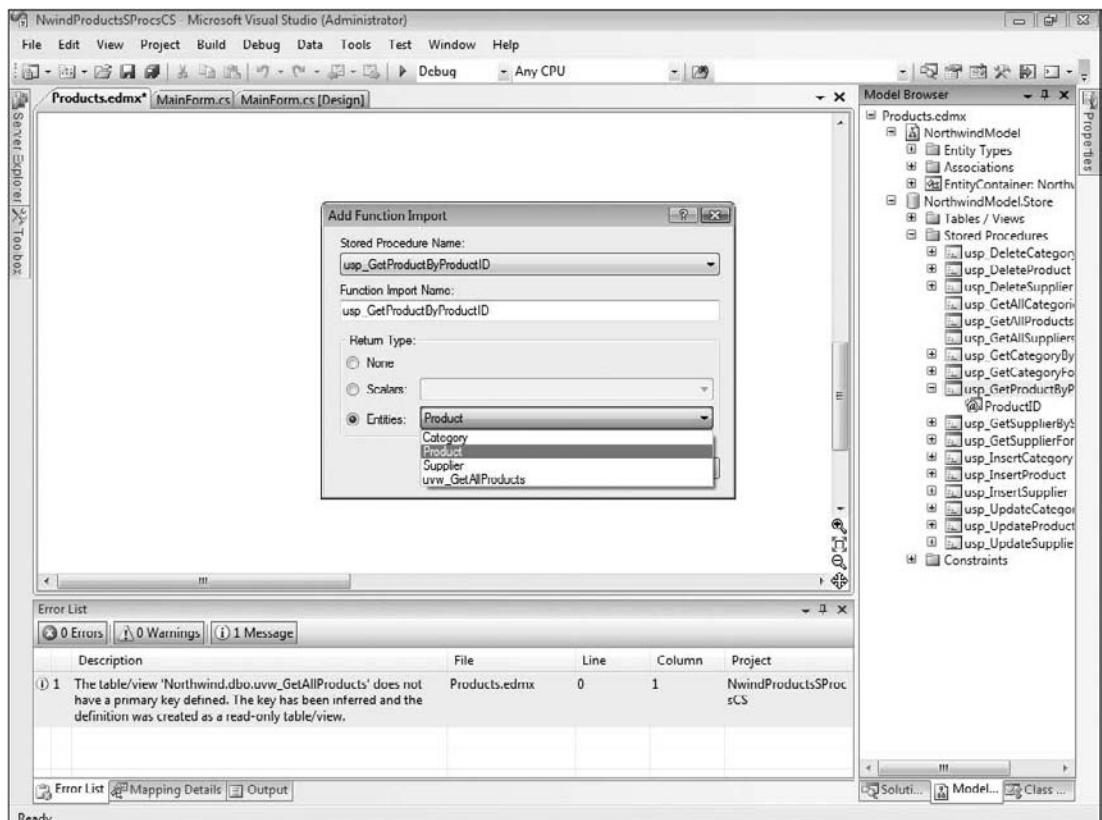


Figure 10-3

## Part IV: Introducing the ADO.NET Entity Framework

---

Click OK to add the following `<FunctionImport>` subgroup to the `<edmx:ConceptualModels>` section's `<Schema>` group:

```
<FunctionImport Name="GetProductByProductID" EntitySet="Products"
    ReturnType="Collection(Self.Product)">
    <Parameter Name="ProductID" Mode="In" Type="Int32" />
</FunctionImport>
```

and the following `<FunctionImportMapping>` element to the `<edmx:Mappings>` section's `<EntityContainerMapping>` group:

```
<FunctionImportMapping FunctionImportAlias="GetProductByProductID"
    FunctionName="NorthwindModel.Store.usp_GetProductByProductID" />
```

*You must create function mappings for all related EntitySets and AssociationSets. After you complete the preceding process for the example's remaining Categories and Suppliers EntitySets, you receive this message in the Error List: "Error 2027: If an entity set or association set includes a function mapping, all related entity and association sets in the entity container must also define function mappings. The following sets require function mappings: Products\_Categories, Products\_Suppliers."*

You'll find more information about the `<FunctionImport>` group and `<FunctionImportMapping>` element in the *Conceptual Models (CSDL Content)* and *Mappings (MSL Content)* sections later in the chapter.

### Insert, Update, and Delete Function Mapping to EntityTypes

EntityTypes populated by views or stored procedures, including `<FunctionImport>`s, are read-only by default and require stored procedure `<Function>`s to perform INSERT, UPDATE, or DELETE operations on the store's tables.

The EDM Designer's Mapping Details pane has a Map Entity to Functions button (in the left margin below the default Map Entity to Tables / Views button), which you can click to display a Functions node with `<Select Insert Function>`, `<Select Update Function>`, and `<Select Delete Function>` subnodes. Select the stored procedure (function) name for each operation from a list that you open with the down-arrow button (see Figure 10-4).

## Chapter 10: Defining Storage, Conceptual, and Mapping Layers

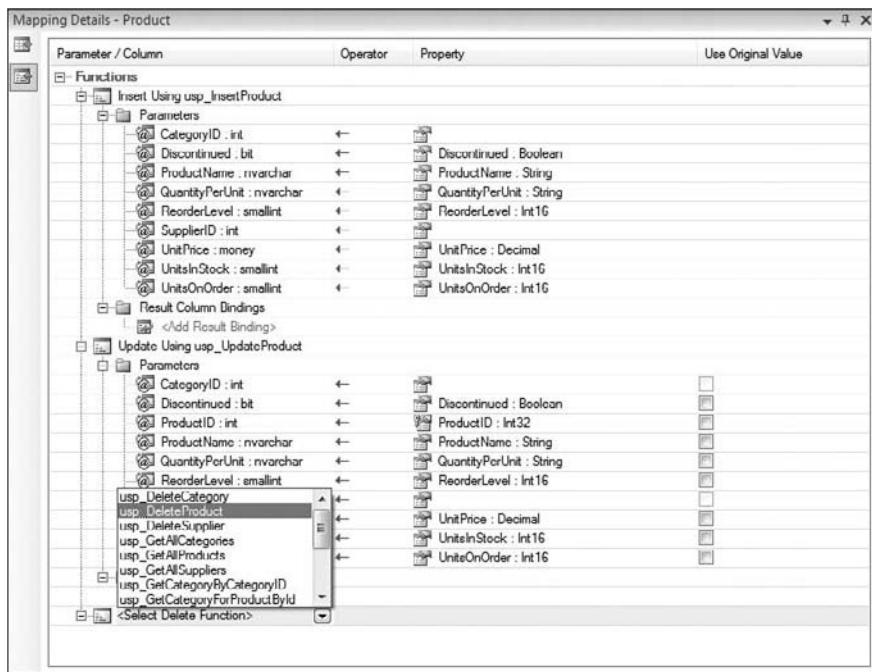


Figure 10-4

The mapper pairs parameters with the entity properties by their names automatically. If there's no match, select a field manually from the Property list to apply the mapping.

Mark the Use Original Value for the primary key's update parameter value so that the updated entity is updated regardless of inadvertent changes to the primary-key value during the update process.

Insert Functions for tables that have an autogenerated primary key require a Result Binding to store the primary key value returned by the `SELECT SCOPE_IDENTITY() AS PrimaryKeyColumn` WHERE `@@ROWCOUNT > 0`; that's required for all INSERT stored procedures. To add the Result Binding, overwrite the `<Add Result Binding>` subnode with the primary key column name, which will attempt a match to the corresponding property name, `ProductID` for the example of Figure 10-4.

After you complete mapping of the Insert, Update, and Delete functions of each entity to the appropriate stored procedures, you'll find error messages such as this for the Products example: "Error 2037: A mapping function bindings specifies a function NorthwindModel.Store.usp\_InsertProduct but does not map the following function parameters: SupplierID, CategoryID."

### Update Stored Procedure Association Mapping in the Designer

The EDM Wizard doesn't automatically map foreign key fields to the corresponding primary keys of related tables. Thus, you must match the pair(s) from a list item in the Properties column. Unlike the Properties list items of the Map Entity to Tables/Views pane, the Map Entity to Function pane's list includes `EntityType.ColumnName` lists of related tables. For this example, select the `Supplier.SupplierID` item to match the `SupplierID` (int) parameter and the `Category.CategoryID` to match the `CategoryID` (int) parameter. After you make the selection, the text changes to `EntitySetName.ColumnName` (see Figure 10-5).

## Part IV: Introducing the ADO.NET Entity Framework

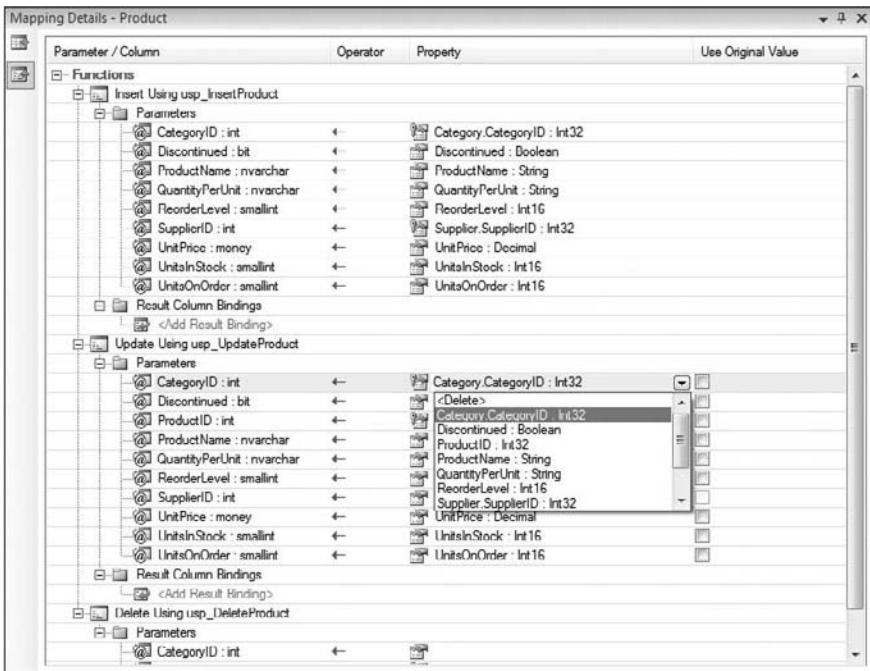


Figure 10-5

You must map the foreign keys of the Insert, Update, and Delete functions to avoid receiving an error message for `usp_DeleteProduct` similar to that in the preceding note. Ordinarily, Delete functions require only the row ID value for selection with the current value. However, you must provide foreign-key mappings for all three functions with a stored procedure such as:

```
CREATE PROCEDURE usp_DeleteProduct (
    @ProductID int,
    @SupplierID int,
    @CategoryID int)
AS
SET NOCOUNT ON

DELETE FROM Products
WHERE ProductID = @ProductID
AND SupplierID = @SupplierID
AND CategoryID = @CategoryID;
```

The mapping enables you to delete associations when the entity is deleted.

*DBAs may balk at providing two or more values to delete the backing row for an entity without value-based concurrency management, but there is no method to avoid this requirement in EF v1.*

## Association Subgroups

Foreign-key constraints, which represent relationships between database tables, generate <Association> subgroups of the <EntityContainer>. Following is the structure of a generic one: many <Association> subgroup for an EntityReference:

```
<Association Name="ForeignKeyConstraintName">
    <End Role="EntitySetName" Type="ModelName.Store.Table1Name"
        Multiplicity="0[..1]" />
    <End Role="EntitySetName" Type="ModelName.Store.Table2Name "
        Multiplicity="*|0[..1]" />
    <ReferentialConstraint>
        <Principal Role="Table1Name">
            <PropertyRef Name="PrimaryKeyFieldName" />
        </Principal>
        <Dependent Role="Table2Name">
            <PropertyRef Name="ForeignKeyFieldName" />
        </Dependent>
    </ReferentialConstraint>
</Association>
```

The <Principal> represents the table with the primary key; the <Dependent> is the foreign-key table.

Following are the <Association> subgroups for the FK\_Products\_Categories and FK\_Products\_Suppliers constraints:

```
<Association Name="FK_Products_Categories">
    <End Role="Categories" Type="NorthwindModel.Store.Categories"
        Multiplicity="0..1" />
    <End Role="Products" Type="NorthwindModel.Store.Products"
        Multiplicity="*" />
    <ReferentialConstraint>
        <Principal Role="Categories">
            <PropertyRef Name="CategoryID" />
        </Principal>
        <Dependent Role="Products">
            <PropertyRef Name="CategoryID" />
        </Dependent>
    </ReferentialConstraint>
</Association>
<Association Name="FK_Products_Suppliers">
    <End Role="Suppliers" Type="NorthwindModel.Store.Suppliers"
        Multiplicity="0..1" />
    <End Role="Products" Type="NorthwindModel.Store.Products"
        Multiplicity="*" />
    <ReferentialConstraint>
        <Principal Role="Suppliers">
            <PropertyRef Name="SupplierID" />
        </Principal>
        <Dependent Role="Products">
            <PropertyRef Name="SupplierID" />
        </Dependent>
    </ReferentialConstraint>
</Association>
```

## Part IV: Introducing the ADO.NET Entity Framework

---

### Population of Associations with Stored Procedures

Populating EntitySets with stored procedures doesn't automatically retrieve and attach Entity instances to populate EntityReferences or EntityCollections. You must write, add Function Imports for, and execute stored procedures to retrieve and attach EntitySets, EntityReferences, or both for the target Entity and its related objects. For example, executing the following T-SQL stored procedures from NwindProductsStoredProcs.sql returns the two EntityReferences for a single Product EntityType:

```
CREATE PROCEDURE usp_GetCategoryForProductById (
    @ProductID int)
AS
SET NOCOUNT ON

SELECT *
FROM Categories AS c JOIN Products AS p
ON p.CategoryID = c.CategoryID
WHERE p.ProductID = @ProductID;

SET NOCOUNT OFF;

CREATE PROCEDURE usp_GetSupplierForProductByID (
    @ProductID int)
AS
SET NOCOUNT ON

SELECT *
FROM Suppliers AS s JOIN Products AS p
ON p.SupplierID = s.SupplierID
WHERE p.ProductID = @ProductID;
```

Creating <FunctionImport> subgroups for stored procedures that populate Associations follows the same pattern as that described in the earlier “Function Imports to Assign Data Retrieval Stored Procedures to an EntitySet” section.

You execute the preceding stored procedures with code like the following to attach the Category and Supplier objects as EntityReference properties to the Product entity and track updates to the associations with the ObjectContext (ctxNwind).

#### C# 3.0

```
ObjectResult<Product> prods = ctxNwind.GetAllProducts();
foreach (Product prod in prods)
{
    prod.CategoryReference
        .Attach(ctxNwind.GetCategoryForProductById(prod.ProductID).First());
    prod.SupplierReference
        .Attach(ctxNwind.GetSupplierForProductById(prod.ProductID).First());
}
```

## VB 9.0

```
Dim prods As ObjectResult(Of Product) = ctxNwind.GetAllProducts()
For Each prod As Product In prods
    prod.CategoryReference _
        .Attach(ctxNwind.GetCategoryForProductById(prod.ProductID).First())
    prod.SupplierReference _
        .Attach(ctxNwind.GetSupplierForProductById(prod.ProductID).First())
End For
```

Using the `EntityNameReference.Attach()` method is mandatory. Attaching Category or Supplier entities as properties of the Product entity with `prod.Category.Attach()` or `prod.Supplier.Attach()` won't create an Association.

*The preceding code is satisfactory for a few Product entities at most because it requires three stored procedure calls for each Product entity. The “Minimizing Stored Procedure Calls and Improving Performance with EF Extensions” topic of Chapter 13.*

## Conceptual Models (CSDL Content)

The `<edmx:ConceptualModels>` group defines the structure of EDM's domain objects in terms of custom CLR classes and standard CLR data types, most of which you create indirectly with the EDM Designer. When you build the project, the CSDL schema generates the custom classes in the `ModelNameModel` namespace of the `ModelName.Designer.cs` or `ModelName.Designer.vb` file.

The `<edmx:ConceptualModels>` group has subgroups that correspond to those of the `<edmx:StorageModels>` group, except that `<Function>` elements or groups, which are `<Schema>` subgroups in SSDL content, move to `<EntityContainer>` subgroups in CSDL content:

```
<edmx:ConceptualModels>
    <Schema Namespace="modelName" Alias="Self"
        xmlns="http://schemas.microsoft.com/ado/2006/04/edm">
        <EntityContainer />
        ...
        <EntityType />
        ...
        <Association />
        ...
    </Schema>
</edmx:ConceptualModels>
```

## The EntityContainer Subgroup

The `<Schema>` group's `<EntityContainer>` subgroup contains the following group hierarchy:

```
<EntityContainer Name="DatabaseNameEntities">
    <EntitySet Name="EntitySetName1" EntityType="modelName.Entity1Type" />
    <EntitySet Name="EntitySetName2" EntityType="modelName.Entity1Type" />
    ...
    <AssociationSet Name="EntitySetName1_EntitySetName2" 
        Association="modelName.EntitySetName1_EntitySetName2">
        <End Role="Role1Name" EntitySet="EntitySetName1" />
```

## Part IV: Introducing the ADO.NET Entity Framework

---

```
<End Role="Role2Name" EntitySet="EntitySet2Name" />
</AssociationSet>
...
[<FunctionImport Name="FunctionImportAlias" EntitySet="EntitySetName"
  ReturnType="Collection(Self.EntityType1)">
  <Parameter Name="ColumnAlias" Mode="InOrOut" Type="CLRTypename" />
</FunctionImport>]
...
</EntityContainer>
```

*<FunctionImport> elements or subgroups appear only for entities populated or updated with stored procedures. Additional <EntitySet> elements would appear in the <EntityContainer> subgroup for derived entities with TPT, TCT, TPH, or MEST inheritance models. As mentioned earlier, Chapter 13 shows you how to implement TPT, TCT, and MEST inheritance models. The later "Implementing Table-per-Hierarchy Inheritance" section shows you how to implement TPH inheritance for the Products entity set by using Discontinued as the discriminator column.*

The CSDL and SSDL <EntityContainer> subgroups for one:one table:type mapping are almost identical. The CSDL <EntityContainer> subgroup includes <Function> elements and subgroups but doesn't include <DefiningQuery> elements for entities based on views.

<EntitySet>, <AssociationSet>, and <FunctionImport> elements or subgroups differ from their SSDL counterparts by definition of their property values in EDM — rather than relational — terminology.

Following is the CSDL content for the <EntityContainer> subgroup with the <Function> elements and subgroups abbreviated:

```
<EntityContainer Name="NorthwindEntities">
  <EntitySet Name="Categories" EntityType="NorthwindModel.Category" />
  <EntitySet Name="Products" EntityType="NorthwindModel.Product" />
  <EntitySet Name="Suppliers" EntityType="NorthwindModel.Supplier" />
  <EntitySet Name="uvw_GetAllProducts"
    EntityType="NorthwindModel.uvw_GetAllProducts" />
  <AssociationSet Name="Products_Categories"
    Association="NorthwindModel.Products_Category">
    <End Role="Categories" EntitySet="Categories" />
    <End Role="Products" EntitySet="Products" />
  </AssociationSet>
  <AssociationSet Name="Products_Suppliers"
    Association="NorthwindModel.Products_Supplier">
    <End Role="Suppliers" EntitySet="Suppliers" />
    <End Role="Products" EntitySet="Products" />
  </AssociationSet>
  <FunctionImport Name="GetAllCategories" EntitySet="Categories"
    ReturnType="Collection(Self.Category)" />
  ...
  <FunctionImport Name="GetProductsBySupplierId" EntitySet="Products"
    ReturnType="Collection(Self.Product)">
    <Parameter Name="SupplierID" Mode="In" Type="Int32" />
  </FunctionImport>
</EntityContainer>
```

## The **EntityType Subgroup**

CSDL <EntityType> groups are identical to their SSDL equivalents except for the substitution of CLR data types for SSDL's database data types and the addition of a <NavigationProperty> element for each foreign key constraint of the underlying table:

```
<EntityType Name="EntityType">
  <Key>
    <PropertyRef Name="PKColumnAlias" />
  </Key>
  <Property Name="PKColumnAlias" Type="ClrNumberOrGuidType" Nullable="false" />
  <Property Name="ColumnAlias" Type="ClrTextType" [Nullable="False"]
            [MaxLength="#"] />
  <Property Name="ColumnAlias" Type="ClrDecimalType" Precision="19" Scale="4" />
  <Property Name="ColumnAlias" Type="ClrNumberType" [Nullable="false"] />
  ...
  <NavigationProperty Name="EntityType"
    Relationship="modelName.AssociationSetName"
    FromRole="Role1Name" ToRole="Role2Name" />
  ...
</EntityType>
```

The <FromRole> represents the property that corresponds to the underlying table foreign-key column and the <ToRole> represents the primary-key property of the related <EntityType>.

Following is the <EntityType> subgroup for the Product type:

```
<EntityType Name="Product">
  <Key>
    <PropertyRef Name="ProductID" />
  </Key>
  <Property Name="ProductID" Type="Int32" Nullable="false" />
  <Property Name="ProductName" Type="String" Nullable="false" MaxLength="40" />
  <Property Name="QuantityPerUnit" Type="String" MaxLength="20" />
  <Property Name="UnitPrice" Type="Decimal" Precision="19" Scale="4" />
  <Property Name="UnitsInStock" Type="Int16" />
  <Property Name="UnitsOnOrder" Type="Int16" />
  <Property Name="ReorderLevel" Type="Int16" />
  <Property Name="Discontinued" Type="Boolean" Nullable="false" />
  <NavigationProperty Name="Category"
    Relationship="NorthwindModel.Products_Category"
    FromRole="Products" ToRole="Categories" />
  <NavigationProperty Name="Supplier"
    Relationship="NorthwindModel.Products_Supplier"
    FromRole="Products" ToRole="Suppliers" />
</EntityType>
```

## Part IV: Introducing the ADO.NET Entity Framework

---

### The Association Subgroup

CSDL <Association> subgroups lack the <ReferentialConstraint> subgroups of their corresponding SSDL subgroups. Otherwise, the only significant difference is removal of the `Store` component from the Type designation:

```
<Association Name="AssociationSetName">
  <End Role="EntityType1Name" Type="ModelName.EntityType1" Multiplicity="0[..1]" />
  <End Role="EntityType2Name" Type="ModelName.EntityType2" Multiplicity="* | 0[..1]" />
</Association>
```

Here's an example of the `Product` EntityType's two many:one associations with the <Category> and <Supplier> EntityType:

```
<Association Name="Products_Category">
  <End Role="Categories" Type="NorthwindModel.Category" Multiplicity="0..1" />
  <End Role="Products" Type="NorthwindModel.Product" Multiplicity="*" />
</Association>
<Association Name="Products_Supplier">
  <End Role="Suppliers" Type="NorthwindModel.Supplier" Multiplicity="0..1" />
  <End Role="Products" Type="NorthwindModel.Product" Multiplicity="*" />
</Association>
```

### Mapping (MSL Content)

The MSL <Mapping> group makes the connection between the domain objects defined at the conceptual layer and the relational tables and foreign key constraints at the physical layer. As you'd expect, the <Mapping> group's <EntityContainerMapping> subgroup, shown here, follows the conceptual model's <EntityContainer> subgroup structure:

```
<edmx:Mappings>
  <Mapping Space="C-S"
    xmlns="urn:schemas-microsoft-com:windows:storage:mapping:CS">
    <EntityContainerMapping StorageEntityContainer="SchemaName"
      CdmEntityContainer="DatabaseNameEntities">
      <EntitySetMapping Name="EntityType1Name">
        ...
        <AssociationSetMapping Name="EntityType1Name_EntityType2Name"
          TypeName="ModelName.EntityType1Name_EntityType2Name"
          StoreEntitySet="EntityType1Name">
          ...
          [<FunctionImportMapping FunctionImportName="FunctionImportAlias"
            FunctionName="ModelName.Store.StoredProcedureName" />]
          ...
        </AssociationSetMapping>
      </EntitySetMapping>
    </EntityContainerMapping>
  </Mapping Space>
</edmx:Mappings>
```

## The EntitySetMapping Subgroup

The default `<EntitySetMapping>` group for entities that use autogenerated dynamic SQL for `INSERT`, `UPDATE`, and `DELETE` operations includes a single `<EntityTypeMapping>` subgroup. Using stored procedures to update the data store adds another `<EntityTypeMapping>` subgroup that contains a `<ModificationFunctionMapping>` group with `<InsertFunction>`, `<UpdateFunction>` and `<DeleteFunction>` subgroups, as shown here:

```
<EntitySetMapping Name="EntitySetName">
    <EntityTypeMapping TypeName="IsTypeOf(ModelName.EntitySetName)">
        <MappingFragment StoreEntitySet="TableName">
            <ScalarProperty Name="PropertyName" Column1Name="ColumnName" />
            ...
        </MappingFragment>
    </EntityTypeMapping>
    [<EntityTypeMapping TypeName="ModelName.EntitySetName">
        <ModificationFunctionMapping>
            <InsertFunction FunctionName="ModelName.Store.InsertProcedureName">
                <ScalarProperty Name="PropertyName" ParameterName="Param1Name" />
                <AssociationEnd AssociationSet="EntitySetName_EntitySet2Name"
                    From="TableName" To="Table2Name">
                    <ScalarProperty Name="ForeignKeyColumnName" ParameterName="Param1Name" />
                </AssociationEnd>
                ...
                <ResultBinding Name="KeyPropertyName" ColumnName="PrimaryKeyColumnName" />
            </InsertFunction>
            <UpdateFunction FunctionName="ModelName.Store.UpdateProcedureName">
                <ScalarProperty Name="PropertyName" ParameterName="Param1Name"
                    Version="{Current|Original}" />
                <AssociationEnd AssociationSet="EntitySetName_EntitySet2Name"
                    From="TableName" To="Table2Name">
                    <ScalarProperty Name="ForeignKeyColumnName" ParameterName="Param1Name" />
                </AssociationEnd>
                ...
            </UpdateFunction>
            <DeleteFunction FunctionName="ModelName.Store.DeleteProcedureName">
                <ScalarProperty Name="PropertyName" ParameterName="Param1Name" />
                <AssociationEnd AssociationSet="EntitySetName_EntitySet2Name"
                    From="TableName" To="Table2Name">
                    <ScalarProperty Name="ForeignKeyColumnName" ParameterName="Param1Name" />
                </AssociationEnd>
                ...
            </DeleteFunction>
        </ModificationFunctionMapping>]
    </EntityTypeMapping>
</EntitySetMapping>
```

Each `<OperationFunction>` group includes an `<AssociationEnd>` element to update the appropriate `AssociationSet` during execution of the stored procedure.

## Part IV: Introducing the ADO.NET Entity Framework

---

Following is the <EntitySetMapping> subgroup content for the Products EntitySet with stored procedures specified for updating:

```
<EntitySetMapping Name="Products">
  <EntityTypeMapping TypeName="IsTypeOf(NorthwindModel.Product)">
    <MappingFragment StoreEntitySet="Products">
      <ScalarProperty Name="ProductID" ColumnName="ProductID" />
      <ScalarProperty Name="ProductName" ColumnName="ProductName" />
      <ScalarProperty Name="QuantityPerUnit" ColumnName="QuantityPerUnit" />
      <ScalarProperty Name="UnitPrice" ColumnName="UnitPrice" />
      <ScalarProperty Name="UnitsInStock" ColumnName="UnitsInStock" />
      <ScalarProperty Name="UnitsOnOrder" ColumnName="UnitsOnOrder" />
      <ScalarProperty Name="ReorderLevel" ColumnName="ReorderLevel" />
      <ScalarProperty Name="Discontinued" ColumnName="Discontinued" />
    </MappingFragment>
  </EntityTypeMapping>
  <EntityTypeMapping TypeName="NorthwindModel.Product">
    <ModificationFunctionMapping>
      <InsertFunction FunctionName="NorthwindModel.Store.usp_InsertProduct">
        <ScalarProperty Name="Discontinued" ParameterName="Discontinued" />
        <ScalarProperty Name="ReorderLevel" ParameterName="ReorderLevel" />
        <ScalarProperty Name="UnitsOnOrder" ParameterName="UnitsOnOrder" />
        <ScalarProperty Name="UnitsInStock" ParameterName="UnitsInStock" />
        <ScalarProperty Name="UnitPrice" ParameterName="UnitPrice" />
        <ScalarProperty Name="QuantityPerUnit" ParameterName="QuantityPerUnit" />
        <ScalarProperty Name="ProductName" ParameterName="ProductName" />
        <AssociationEnd AssociationSet="Products_Suppliers"
          From="Products" To="Suppliers">
          <ScalarProperty Name="SupplierID" ParameterName="SupplierID" />
        </AssociationEnd>
        <AssociationEnd AssociationSet="Products_Categories"
          From="Products" To="Categories">
          <ScalarProperty Name="CategoryID" ParameterName="CategoryID" />
        </AssociationEnd>
        <ResultBinding Name="ProductID" ColumnName="ProductID" />
      </InsertFunction>
      <UpdateFunction FunctionName="NorthwindModel.Store.usp_UpdateProduct">
        <ScalarProperty Name="Discontinued" ParameterName="Discontinued"
          Version="Current" />
        <ScalarProperty Name="ReorderLevel" ParameterName="ReorderLevel"
          Version="Current" />
        <ScalarProperty Name="UnitsOnOrder" ParameterName="UnitsOnOrder"
          Version="Current" />
        <ScalarProperty Name="UnitsInStock" ParameterName="UnitsInStock"
          Version="Current" />
        <ScalarProperty Name="UnitPrice" ParameterName="UnitPrice"
          Version="Current" />
        <ScalarProperty Name="QuantityPerUnit" ParameterName="QuantityPerUnit"
          Version="Current" />
        <ScalarProperty Name="ProductName" ParameterName="ProductName"
          Version="Current" />
      </UpdateFunction>
    </ModificationFunctionMapping>
  </EntityTypeMapping>
</EntitySetMapping>
```

## Chapter 10: Defining Storage, Conceptual, and Mapping Layers

---

```
<ScalarProperty Name="ProductID" ParameterName="ProductID"
    Version="Current" />
<AssociationEnd AssociationSet="Products_Suppliers"
    From="Products" To="Suppliers">
    <ScalarProperty Name="SupplierID" ParameterName="SupplierID"
        Version="Current" />
</AssociationEnd>
<AssociationEnd AssociationSet="Products_Categories"
    From="Products" To="Categories">
    <ScalarProperty Name="CategoryID" ParameterName="CategoryID"
        Version="Current" />
</AssociationEnd>
</UpdateFunction>
<DeleteFunction FunctionName="NorthwindModel.Store.usp_DeleteProduct">
    <ScalarProperty Name="ProductID" ParameterName="ProductID" />
    <AssociationEnd AssociationSet="Products_Categories"
        From="Products" To="Categories">
        <ScalarProperty Name="CategoryID" ParameterName="CategoryID" />
    </AssociationEnd>
    <AssociationEnd AssociationSet="Products_Suppliers"
        From="Products" To="Suppliers">
        <ScalarProperty Name="SupplierID" ParameterName="SupplierID" />
    </AssociationEnd>
</DeleteFunction>
</ModificationFunctionMapping>
</EntityTypeMapping>
</EntitySetMapping>
```

Specifying the `Version` attribute's value is only significant when optimistic concurrency management is specified and the UPDATE stored procedure includes `ColumnName_OriginalValue` or similar parameters.

### The **AssociationSetMapping Subgroup**

`<AssociationSetMapping>` subgroups involve two entity sets within the `EntityContainer`, so they must one of its subgroups. `<AssociationSetMapping>` subgroups introduce a `<Condition>` expression that specifies that the association is valid for non-null foreign-key values only:

```
<AssociationSetMapping Name="EntitySetName_EntitySetName"
    TypeName="modelName.EntitySetName_EntitySetName" StoreEntitySet="Table1Name">
    <EndProperty Name="EntitySetName">
        <ScalarProperty Name="EntityType2Key" ColumnName="Table2PrimaryKey" />
    </EndProperty>
    <EndProperty Name=" EntitySetName">
        <ScalarProperty Name="EntityType1PrimaryKey" ColumnName="Table1PrimaryKey" />
    </EndProperty>
    <Condition ColumnName="EntityType1ForeignKey" IsNull="false" />
</AssociationSetMapping>
```

## Part IV: Introducing the ADO.NET Entity Framework

---

Here are the two `<AssociationSetMapping>` subgroups for the `Products` EntitySet:

```
<AssociationSetMapping Name="Products_Categories"
    TypeName="NorthwindModel.Products_Category" StoreEntitySet="Products">
    <EndProperty Name="Categories">
        <ScalarProperty Name="CategoryID" ColumnName="CategoryID" />
    </EndProperty>
    <EndProperty Name="Products">
        <ScalarProperty Name="ProductID" ColumnName="ProductID" />
    </EndProperty>
    <Condition ColumnName="CategoryID" IsNull="false" />
</AssociationSetMapping>
<AssociationSetMapping Name="Products_Suppliers"
    TypeName="NorthwindModel.Products_Supplier" StoreEntitySet="Products">
    <EndProperty Name="Suppliers">
        <ScalarProperty Name="SupplierID" ColumnName="SupplierID" />
    </EndProperty>
    <EndProperty Name="Products">
        <ScalarProperty Name="ProductID" ColumnName="ProductID" />
    </EndProperty>
    <Condition ColumnName="SupplierID" IsNull="false" />
</AssociationSetMapping>
```

### **The `FunctionImportMapping` Element**

`<FunctionImportMapping>` elements apply only to data retrieval stored procedures; these elements map the database's stored procedure name to the alias you specify in the Function Import Mapping dialog:

```
<FunctionImportMapping FunctionImportName="FunctionImportAlias"
    FunctionName="ModelName.Store.ProcedureName" />
```

*It's a common practice to remove stored procedure prefixes to create the alias.*

Following are the data retrieval stored procedures used by this chapter's sample projects:

```
<FunctionImportMapping FunctionImportName="GetAllCategories"
    FunctionName="NorthwindModel.Store.usp_GetAllCategories" />
<FunctionImportMapping FunctionImportName="GetAllProducts"
    FunctionName="NorthwindModel.Store.usp_GetAllProducts" />
<FunctionImportMapping FunctionImportName="GetAllSuppliers"
    FunctionName="NorthwindModel.Store.usp_GetAllSuppliers" />
<FunctionImportMapping FunctionImportName="GetCategoryForProductById"
    FunctionName="NorthwindModel.Store.usp_GetCategoryForProductById" />
<FunctionImportMapping FunctionImportName="GetSupplierForProductById"
    FunctionName="NorthwindModel.Store.usp_GetSupplierForProductByID" />
```

*Inconsistencies in the `<Mapping>` group are the most common source of error messages.*

# Implementing Table-per-Hierarchy Inheritance

Table-per-hierarchy (TPH) is the simplest inheritance model to implement with the EDM Designer. The TPH model incorporates all rows and columns for the base and derived type(s) in a single data store table. Values in a specified column, called the *discriminator column*, determine whether the row belongs to the base type or a particular derived type.

The Products entity and its underlying Products table is a logical candidate for demonstrating the TPH inheritance model because it contains a good choice for the discriminator column: Discontinued. This example uses the EDM Designer to create a new DiscontinuedProducts entity for Products whose Discontinued property is true. An added DateDiscontinued column demonstrates a column whose values appear in the derived type but are NULL in the base type. The process for adding the derived type starts with a sample project that has Product, Category and Supplier entities: NwindProductsTSqlCS.sln or NwindProductsTSqlVB.sln in your \WROX\ADONET\Chapter10\CS or ... \VB folder.

*A peculiarity of TPH entity inheritance — especially to data-oriented developers — is that derived entities don't have EntitySets; this example has no DiscontinuedProducts EntitySet. The backing rows for derived products remain in the base class's table and are identified by their discriminator column property.*

*In addition, the EF provides no means to update the discriminator column value. Thus, there is no object-oriented method to denote an entity from the Product to DiscontinuedProduct type because doing this would require deleting Product entity and adding its set of property values to with a modified discriminator value as a DiscontinuedProduct entity, which would insert a new row with a new autogenerated ProductID cell and ProductID key value. You can overcome this issue by adding an update trigger to the table that responds to a change from NULL to a date value by setting the Discontinued column's bit data type for the row to 0. An alternative is to execute a stored procedure from a handler for the DateDiscontinued property's PropertyChanged event or a stored procedure to change the DateDiscontinued and Discontinued column values from a button\_click event.*

## Specifying the Discriminator Column and Creating a Derived Class

Start by using Server Explorer to add a nullable DateDiscontinued column of the datetime data type to your Northwind instance's Products table and then right-click the Model Browser pane and select Update Model from Database to add the column and property to the base entity. The DateDiscontinued property is a member of the Product entity, which enables you to change it to the current date and, optionally, time to enable an external update to the Discontinued column.

To specify the Discontinued column as the discriminator, select the Product widget, click the Mapping Details tab to open the Mapping Details pane, click the Discontinued row's Value/Property column to open the mapping drop-down list and select <Delete> to remove the mapping. (You can't select a mapped column as the discriminator.) Click the <Add a Condition> node to open the list of unmapped columns and select Discontinued to change the node name to When Discontinued, add = as the Operator and set the Value/Property column false because the entity type will be DiscontinuedProduct when the Discontinued property value is true (see Figure 10-6). Press Ctrl+S to save your changes.

## Part IV: Introducing the ADO.NET Entity Framework

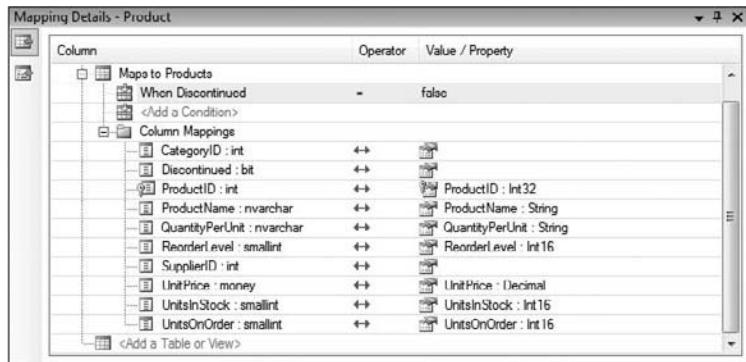


Figure 10-6

To add the `DisconnectedProduct` derived type, right-click an empty region of the EDM Designer pane and choose Add, Entity to open the New Entity dialog. Type `DisconnectedProduct` as the Entity Name and select `Product` as the Base Type to display `Products` as the Entity Set (see Figure 10-7). Click OK to add the derived type to the Designer window.



Figure 10-7

With the `DisconnectedProduct` widget selected, click the Mapping Details tab to open the Mapping Details pane, click the `<Add Table or View>` element to open a drop-down list, and select `Products`. Click the `<Add a Condition>` node to open the list of unmapped columns, select `Discontinued`, and set the `true` to assign the `DisconnectedProduct` type (see Figure 10-8).

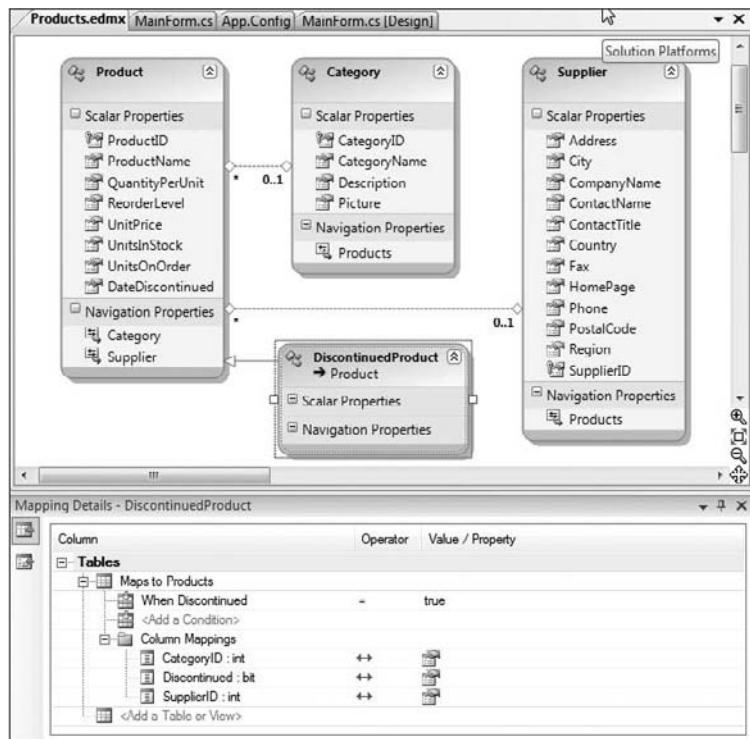


Figure 10-8

If the Discontinued property appears in the Product widget's list, delete it, then press Ctrl+S to save your changes and press F5 to build and run your project. Press Shift+F5 to halt execution and open the Errors list to verify that your instance has no errors.

If you encounter errors, try to correct the error by modifying the entities in the EDM Designer. If you can't correct the error graphically, open the Northwind.edmx file in the XML Editor, open the Error List, if necessary, double-click the error entry to move to the offending line, correct the error (if you can), save the changes, and press F5 to build and run the project again.

*The NwindProductsTphCS.sln or NwindProductsTphVB.sln in your \WROX\ADONET\Chapter10\CS or . . . \VB folder contains the final version of the TPH project with the source code for the following section.*

### Querying Base and Derived Classes

TPH types and derived types have some surprises in store for you when you write queries that you expect to return Product or DiscontinuedProduct entities but not both. Figure 10-9 shows the UI of the NwindProductsTphCS project displaying the results for five of six tests of a few simple queries against entities based on the original Northwind Products table, which has 77 products, of which 8 are discontinued.

## Part IV: Introducing the ADO.NET Entity Framework

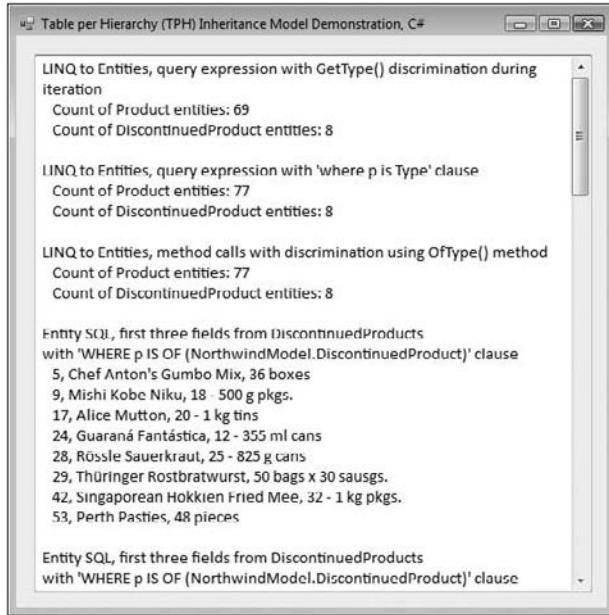


Figure 10-9

Test 1 is the only one of the first three tests that returns the correct count of `Product` entities (69) and `DiscontinuedProduct` entities (8) from these two `EntitySets`. This test uses simple iteration over the `IQueryable<Product>` sequence returned by a simple LINQ to Entities query with the `typeof(TypeName)` and `GetType(TypeName)` tests. The following snippet from the project's `MainForm_Load` event handler performs the test and adds a current `DiscontinuedDate` value to `DiscontinuedProducts`:

### C# 3.0

```
int prod = 0;
int disc = 0;
NorthwindEntities ctxNwind = new NorthwindEntities();
var query = from p in ctxNwind.Products select p;
foreach (var p in query)
{
    Type t = p.GetType();
    if (t == typeof(Product))
    {
        prod += 1;
    }
    else if (t == typeof(DiscontinuedProduct))
    {
        disc += 1;
        p.DateDiscontinued = DateTime.Today;
    }
}
ctxNwind.SaveChanges();
```

### VB 9.0

```
Dim prod As Integer = 0
Dim disc As Integer = 0
Dim ctxNwind As New NorthwindEntities()
Dim query = From p In ctxNwind.Products Select p
For Each p In query
    Dim t As Type = p.GetType()
    If t Is GetType(Product) Then
        prod += 1
    ElseIf t Is GetType(DiscontinuedProduct) Then
        disc += 1
        p.DateDiscontinued = DateTime.Today
    End If
Next p
ctxNwind.SaveChanges()
```

*Online help describes the C# `typeof(typename)` method as “Used to obtain the `System.Type` object for a type” and the VB `GetType(typename)` operator as “The `GetType` operator returns the `Type` object for the specified `typename`. You can pass the name of any defined type in `typename`.” Online help also advises: “To obtain the run-time type of an expression, you can use the .NET Framework method `GetType`, as in the following example: `int i = 0; System.Type type = i.GetType();`”*

### Incorrect Results from `is`, `Is`, `TypeOf`, and `OfType()` Operators

Test 2 applies the `Count()` method to a pair of simple query expressions that use the `is/Is` keyword. As you can see if Figure 9, Test 2 counts 77 Products and 8 DiscontinuedProducts.

### C# 3.0

```
prod = (from p in ctxNwind.Products
        where p is Product
        select p).Count();
disc = (from p in ctxNwind.Products
        where p is DiscontinuedProduct
        select p).Count();
```

### VB 9.0

```
prod = (From p In ctxNwind.Products _
        Where TypeOf p Is Product _
        Select p).Count()
disc = (From p In ctxNwind.Products _
        Where TypeOf p Is DiscontinuedProduct _
        Select p).Count()
```

The C# `is` expression returns `true` if an object is compatible with a given type; `DiscontinuedProduct` is compatible with its derived type `Product`, so 77 is the correct `Product` count. However, the VB version of `Is` compares two object reference variables. Online help says “If `object1` and `object2` both refer to the same object, `result` is `True`; if they do not, `result` is `False`.” Therefore, the VB `Product` count should be 69.

## Part IV: Introducing the ADO.NET Entity Framework

---

Test 3 returns the same incorrect Product counts with unambiguous `TypeOf<GenericType>` operators; both of the following snippets return 77 as the Product count and 8 as the DiscontinuedProduct count:

### C# 3.0

```
prod = ctxNwind.Products.OfType<Product>().Count();
disc = ctxNwind.Products.OfType<DiscontinuedProduct>().Count();
```

### VB 9.0

```
prod = ctxNwind.Products.OfType(Of Product)().Count()
disc = ctxNwind.Products.OfType(Of DiscontinuedProduct)().Count()
```

## Type Discrimination in Entity SQL Queries

Entity SQL uses the `IS OF (ModelName.TypeName)` operator/function combination as a WHERE clause condition to apply a specific type restriction to the set returned by a `SELECT VALUE` query. For example, the following snippets return and iterate the 8-row resultset of `DiscontinuedProducts` shown as Test 4 in Figure 10-9:

### C# 3.0

```
using (EntityConnection cnn = new EntityConnection("Name=NorthwindEntities"))
{
    cnn.Open();
    try
    {
        string sql = "SELECT VALUE p FROM NorthwindEntities.Products AS p " +
                     "WHERE p IS OF (NorthwindModel.DiscontinuedProduct)";

        using (EntityCommand cmd = new EntityCommand(sql, cnn))
        {
            EntityDataReader reader =
                cmd.ExecuteReader(CommandBehavior.SequentialAccess);
            while (reader.Read())
            {
                txtResult.Text += String.Format(" {0}, {1}, {2}",
                                                reader[0], reader[1], reader[2]) + "\r\n";
            }
            reader.Close();
        }
        cnn.Close();
    }
}
```

### VB 9.0

```
Using cnn As New EntityConnection("Name=NorthwindEntities")
    cnn.Open()
    Try
        Dim sql As String = "SELECT VALUE " & _
                            "TREAT(p AS NorthwindModel.DiscontinuedProduct) " & _
                            "FROM NorthwindEntities.Products AS p " & _
                            "WHERE p IS OF (NorthwindModel.DiscontinuedProduct)"

        Using cmd As New EntityCommand(sql, cnn)
            Dim reader As EntityDataReader =
```

# Chapter 10: Defining Storage, Conceptual, and Mapping Layers

---

```
cmd.ExecuteReader(CommandBehavior.SequentialAccess)
Do While reader.Read()
    txtResult.Text += String.Format(" {0}, {1}, {2}", _
        reader(0), reader(1), reader(2)) & Constants.vbCrLf
Loop
reader.Close()
End Using
cnn.Close()
Catch ex As Exception
    Console.WriteLine(ex.Message)
End Try
End Using
```

Test 5, which omits the WHERE p IS OF (NorthwindModel.DiscontinuedProduct) or substitutes the WHERE p IS OF (NorthwindModel.Product) criterion returns a total of 77 rows; 8 rows contain data and 69 rows are empty.

## **Disambiguate Derived Object Types with an Abstract Base Type**

You can eliminate ambiguities in query type discrimination by defining a new derived type, such as ActiveProduct, to represent remaining Product base class entities. Doing this enables queries to explicitly distinguish ActiveProduct from DiscontinuedProduct entities; in the process, the Product base type loses all its mappings and becomes an *abstract base type*.

Creating an ActiveProduct derived class follows the same process as creating the DiscontinuedProduct class, except for the Entity Name and the Condition, which becomes the same as the Product entity's When Discontinued = false value. Then Open the Product entity's Properties sheet, change the Abstract property from False to True, open the Mapping Details pane and remove the Condition.

You must alter the code that formerly attempted to perform the equivalent of a Products EXCEPT DiscontinuedProducts union. For example, Test 1's code must change the test for the Product type to ActiveProduct, as shown here in bold type:

### **C# 3.0**

```
int prod = 0;
int disc = 0;
NorthwindEntities ctxNwind = new NorthwindEntities();
var query = from p in ctxNwind.Products select p;
foreach (var p in query)
{
    Type t = p.GetType();
    if (t == typeof(ActiveProduct))
    {
        prod += 1;
    }
    else if (t == typeof(DiscontinuedProduct))
    {
        disc += 1;
        p.DateDiscontinued = DateTime.Today;
    }
}
ctxNwind.SaveChanges();
```

## Part IV: Introducing the ADO.NET Entity Framework

### VB 9.0

```
Dim prod As Integer = 0
Dim disc As Integer = 0
Dim ctxNwind As New NorthwindEntities()
Dim query = From p In ctxNwind.Products Select p
For Each p In query
    Dim t As Type = p.GetType()
    If t Is GetType(ActiveProduct) Then
        prod += 1
    ElseIf t Is GetType(DiscontinuedProduct) Then
        disc += 1
        p.DateDiscontinued = DateTime.Today
    End If
Next p
ctxNwind.SaveChanges()
```

After you complete the changes for the remainder of the code, the results for the first five tests appears as shown in Figure 10-10, which is the UI for the NwindProductsTph2CS.sln or NwindProductsTph2VB.sln sample projects.

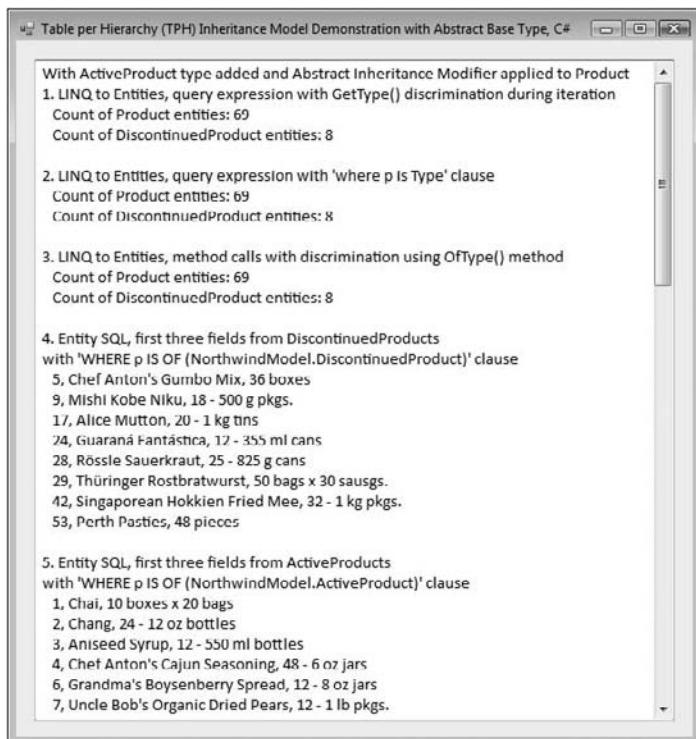


Figure 10-10

All tests in the NwindProductsTph2CS.sln or NwindProductsTph2VB.sln sample projects return the correct count.

## Traversing the MetadataWorkspace

The `MetadataWorkspace` caches the settings you specify in the `ModelName.edmx` file when you build and run an EF project. This operation consumes substantial disk I/O and memory resources, so expect a performance hit when you start up a project that includes a large EDM. Using the AdventureWorks as an example, the `AW.edmx` file contains 7,054 lines without including views or stored procedures in the EDM. The `AW.Designer.cs` file's `AdventureWorksEntities` class defines 66 partial `EntityType` classes and 91 `Associations` in 17,392 lines of generated code. Finding the `Members` of an `EntityType` by querying the `MetadataWorkspace` is much faster than using reflection.

*This section's sample projects require an instance of the AdventureWorks database for SQL Server 2005 SP2a (`AdventureWorksDB.msi`) or SQL Server 2008 (`AdventureWorks2008.msi`), which you can download from the CodePlex site (<http://www.codeplex.com/MSFTDBProdSamples>).*

You access items in the `MetadataWorkspace` by opening an `EntityConnection` and invoking the `GetMetadataWorkspace()` method on it. Then execute a LINQ to Entities query over the `MetadataWorkspace` instance to return the collection you want by invoking the appropriate `GetEntityContainer()`, `GetItems<T>()`, or `GetFunctions<T>()` method from the `DataSpace` enumeration member you supply as an argument. The following table provides a description for each member of the `DataSpace` enumeration.

DataSpace Member	Represents
<code>CSpace</code>	The conceptual model (CSDL)
<code>CSSpace</code>	The mapping layer between the conceptual model and the storage model (MSL)
<code>OCSpace</code>	The mapping layer between the object model and the conceptual model
<code>OSpace</code>	The object model (classes in <code>ModelName.Designer.cs</code> or <code>.vb</code> )
<code>SSpace</code>	The storage model (SSDL)

## Part IV: Introducing the ADO.NET Entity Framework

---

The following snippet, which is part of the \Wrox\ADONET\Chapter10\CS\AdventureWorksMetadataCS.sln project, enumerates the `EntitySet` collections and the `RelationshipType` and `RelationshipEnd` items from the `AdventureWorksModel`:

### C# 3.0

```
using (EntityConnection conn =
    new EntityConnection("Name = AdventureWorksEntities"))
{
    StringBuilder sb = new StringBuilder();
    MetadataWorkspace metaWork = conn.GetMetadataWorkspace();
    var entitySets = from eContainer
        in metaWork.GetItems<EntityContainer>(DataSpace.CSpace)
        from eSet in eContainer.BaseEntitySets
        where eSet is EntitySet
        orderby eSet.Name
        select eSet;
    foreach (EntitySet eSet in entitySets)
    {
        sb.Append(String.Format("EntitySet '{0}'; EntityType '{1}'",
            eSet.Name, eSet.ElementType.Name) + "\r\n");
        entities += 1;
        foreach (RelationshipType relType in
            metaWork.GetItems<RelationshipType>(DataSpace.CSpace))
        {
            foreach (RelationshipEndMember end in
                relType.RelationshipEndMembers)
            {
                // Strings materialized for readability
                string endName = end.TypeUsage.EdmType.Name;
                // reference[Namespace.Element]
                string elementName = eSet.ElementType.FullName;
                string endNameFix = endName.Substring(10, endName.Length - 11);
                if (endNameFix == elementName)
                    sb.Append("    Association End' " + relType.Name + "'\r\n");
                break;
            }
        }
    }
    txtResult.Text = sb.ToString();
}
```

Figure 10-11 shows the AdventureWorksMetadataVB project displaying the first 10 of the 66 `EntitySets` in the conceptual layer.

## Chapter 10: Defining Storage, Conceptual, and Mapping Layers

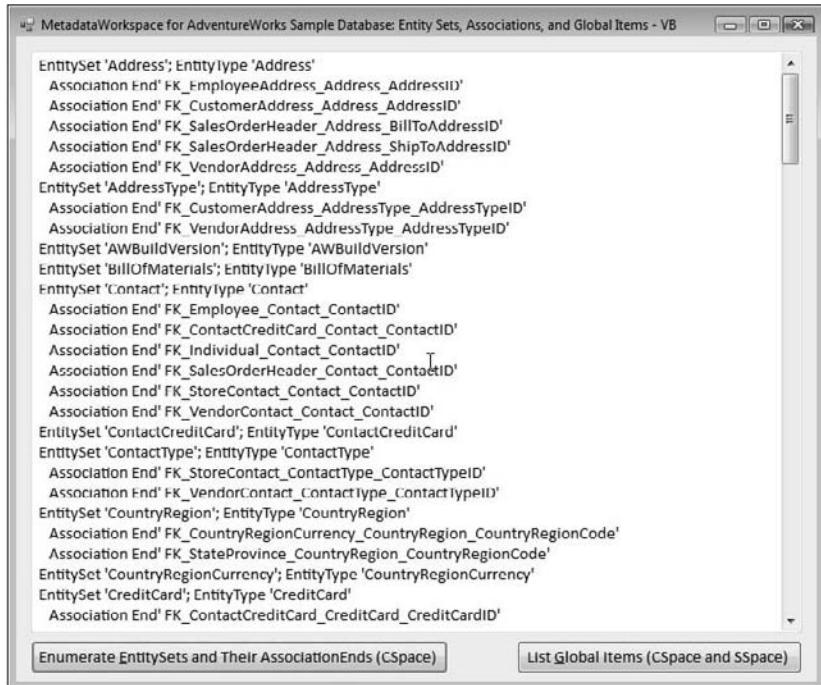


Figure 10-11

Following is the VB version:

### VB 9.0

```
Using conn As New EntityConnection("Name = AdventureWorksEntities")
    Dim sb As New StringBuilder()
    Dim metaWork As MetadataWorkspace = conn.GetMetadataWorkspace()
    Dim entitySets = From eContainer ) _
        In metaWork.GetItems(Of EntityContainer)(DataSpace.CSpace) _
        From eSet In eContainer.BaseEntitySets _
        Where TypeOf eSet Is EntitySet _
        Order By eSet.Name _
        Select eSet

    Dim entities As Integer = 0
    Dim associations As Integer = 0
    For Each eSet As EntitySet In entitySets
        sb.Append(String.Format("EntitySet '{0}'; EntityType '{1}'", _
            eSet.Name, eSet.ElementType.Name) & Constants.vbCrLf)
        entities += 1
        For Each relType As RelationshipType _
            In metaWork.GetItems(Of RelationshipType)(DataSpace.CSpace)
            For Each [end] As RelationshipEndMember _
                In relType.RelationshipEndMembers
```

## Part IV: Introducing the ADO.NET Entity Framework

```
' Strings materialized for readability
Dim endName As String = [end].TypeUsage.EdmType.Name
' reference[Namespace.Element]
Dim elementName As String = eSet.ElementType.FullName
Dim endNameFix As String = endName.Substring(10, _
    endName.Length - 11)
If endNameFix = elementName Then
    sb.Append("    Association End' " & relType.Name & _
        "' " & Constants.vbCrLf)
End If
Exit For
Next [end]
Next relType
Next eSet
txtResult.Text = sb.ToString()
End Using
```

Alternatively, you can browse the entire conceptual and storage layers to retrieve PrimitiveTypes (.NET and data provider data types), Functions (EDM canonical and data-provider-specific), EntityContainers, EntityTypes, and AssociationTypes for both CSpace and SSpace as a flat list with nested Member and Association.End entries with the code executed by the List Global Items button's event handler in the AdventureWorksMetadataCS and VB projects. Figure 10-12 shows a very small part of the 2,377 items in the list.

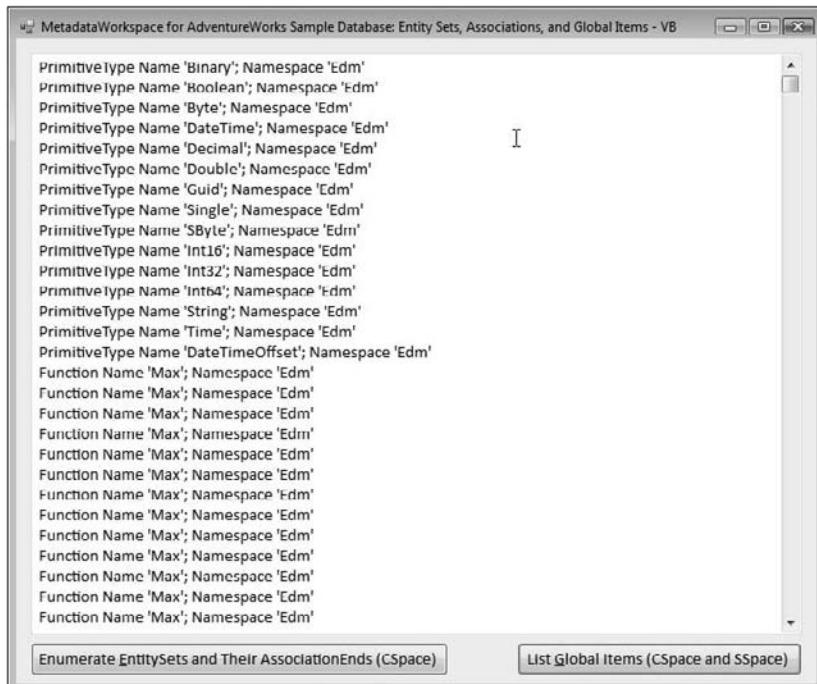


Figure 10-12

## Chapter 10: Defining Storage, Conceptual, and Mapping Layers

---

*Code in the List Global Items button's event handler is based on the sample application from the "GlobalItem Class" EF online help topic modified to display nested Member and Association.End entries.*

## Summary

The `ModelName.edmx` file is EF's modeling centerpiece; the EDM is totally dependent on this file in design mode and the `ModelName.csdl`, `ModelName.msl`, and `ModelName.ssdl` files it generates at runtime. The EDM Designer handles most model development tasks with aplomb but, if a problem occurs, you'll probably need to edit the `ModelName.edmx` file. The introduction pointed out that you should be at least conversant with the files structure and contents.

The chapter began with an analysis of the `ModelName.edmx` file's Storage Models (SSDL Content) section, which represents the persistence store, because the EF version on which this chapter is primarily based supports creating data models from the chosen database, Northwind running on an SQL Server 2005 or 2008 Express instance. Substituting stored procedures for autogenerated SQL batches for CRUD operations primarily involves the SSDL Content, so the chapter covered the topic in the physical layer context. Chapter 13 shows you how to execute stored procedures and demonstrates their advantages and disadvantages in production applications.

The "Conceptual Models (CSDL Content)" section defined the EntityTypes that appear on the EDM Designer's surface and the Mapping (MSL Content) connects the CSDL content to the SSDL content. Thus the presentation of this information was in a different order (CSDL before MSL) than you might have expected from the order of the three layers shown in the EF dataflow diagram at the beginning of the chapter.

Flexible inheritance mapping is one of the EDM's strong points. The chapter described how to implement the most common EF inheritance model and LINQ to SQL's only inheritance mode: table-per-hierarchy (TPH). The "Implementing Table-per-Hierarchy Inheritance" section described how to derive a `DiscontinuedProduct` type from the `Product` base `EntityType`, examined the ambiguities involved in querying the `Product` type, and demonstrated the advantages of deriving an additional `ActiveProduct` type to enable retiring the `Product` type to an abstract class.

The "Traversing the MetadataWorkspace" section described how to enumerate the runtime representation of the CSDL content to obtain read-only information from the `MetadataWorkspace` about `EntitySets`, `Associations` and their `Members`, as well as `EntityContainers`, `EntityType`s, and `AssociationType`s in the SSDL content (`SSpace`).



# 11

## Introducing Entity SQL

Most O/RM tools have a proprietary query language that's based on SQL with object-oriented extensions or an object query language (OQL). For example, Hibernate and NHibernate use Hibernate Query Language (HQL), LLBLGen Pro generates dynamic SQL, and Genome uses an OQL with a syntax similar to C#. The primary incentives for providing a proprietary query language are to enable querying persistence stores from multiple RDBMS vendors and, in most cases, to handle nonrelational features such as collections, associations, many:many relationships, and inheritance hierarchies. Product-specific LINQ implementations can eliminate the need for developers to learn multiple query languages. The three example O/RM tools have their own LINQ variations, which this book calls LINQ to NHibernate, LINQ to LLBLGen, and LINQ to Genome.

The Entity Framework's proprietary SQL dialect is Entity SQL, often called *eSQL*. *eSQL* queries operate at the Entity Data Model's conceptual layer and execute against the `EntityClient` object to deliver a high-performance `EntityDataReader`. The `EntityDataReader` returns `DbDataRecords` that implement `IExtendedDataRecord` to support hierarchies for entities with associations. Alternatively, you can execute an `eSQL ObjectQuery` against the Object Services layer to return an `ObjectQueryResult` collection of `EntityType` or anonymous type instances. Chapter 12 covers executing *eSQL* queries against the Object Services layer.

*eSQL*'s syntax is similar to T-SQL, but *eSQL* has many reserved words that are specific to support for nonrelational features such as associations, references and inheritance. For example, `NAVIGATE()` traverses associations, `CREATeref()` creates a reference from an `EntityType` instance, `DEREF()` does the opposite, and `OFTYPE()` lets you filter by base or derived `EntityType`. *eSQL* uses *item* and *collection* expressions; source expressions can be *literals*, *parameters*, or *nested expressions*. The use of expressions to create queries means that *eSQL* is *composable*. Composability means that query components can be assembled in particular combinations that best suit the query's objectives.

## Part IV: Introducing the ADO.NET Entity Framework

---

eSQL v1 is a querying (retrieval) language only and doesn't support SQL's Data Manipulation Language (DML, `INSERT`, `UPDATE`, and `DELETE`) or Data Definition Language (DDL, `CREATE TABLE`, `ALTER TABLE`, `DROP TABLE`, and so on). V1 requires direct execution of DML and DDL statements in the persistence store's native SQL dialect by ADO.NET `dbConnection` and `dbCommand` objects.

Alternatively, inserting, updating, or deleting entity instances at the Object Services layer can generate DML operations, as demonstrated in Chapter 12.

## Using the eSqlBlast Entity SQL Query Utility

The `eSqlBlast.sln` project in the `\WROX\ADONET\Chapter11` folder is very useful utility for displaying the entities, complex types, and associations defined by an arbitrary `ModelName.csdl` file as well as the contents of `SqlDataRecords` from eSQL queries against SQL Server data stores. `eSqlBlast` lets you execute eSQL queries against the `EntityClient` and observe their results without writing your own test harness. Most of this chapter's examples use a slightly modified `eSqlBlast` version to demonstrate eSQL-specific query syntax.

Zlatko Michaelev, an ADO.NET program manager in the Data Programmability Runtime group, wrote the `eSqlBlast` program, which has Windows, Web, and command-line versions, and placed the source code updated for VS 2008 SP1 on the MSDN Code Gallery (<http://code.msdn.microsoft.com/esql>) for distribution under the Microsoft Permissive License (Ms-PL). The version included with this book's sample code has the 8.5-pt Microsoft Sans-Serif font changed to 11-point Calibri for more readable screen captures. It also has altered default values to simplify generating a connection string for the Northwind sample database attached to an SQL Server 2005+ Express instance and an added `Northwind.edmx` file to generate three mapping files.

The following four sections describe `eSqlBlast`'s Connection, Model, Query, and Results pages.

### Connection Page

The Connection page requires a Provider designation, Provider Connection string to the persistence store — the Northwind sample database running on an SQL Server 2005+ [Express] instance — and the path to the ...`\WinShell\bin\debug` folder, which contains the metadata files (`Northwind.csdl`, `Northwind.msl`, and `Northwind.ssdl` for this example). Figure 11-1 shows the Connection page with default values unchanged.

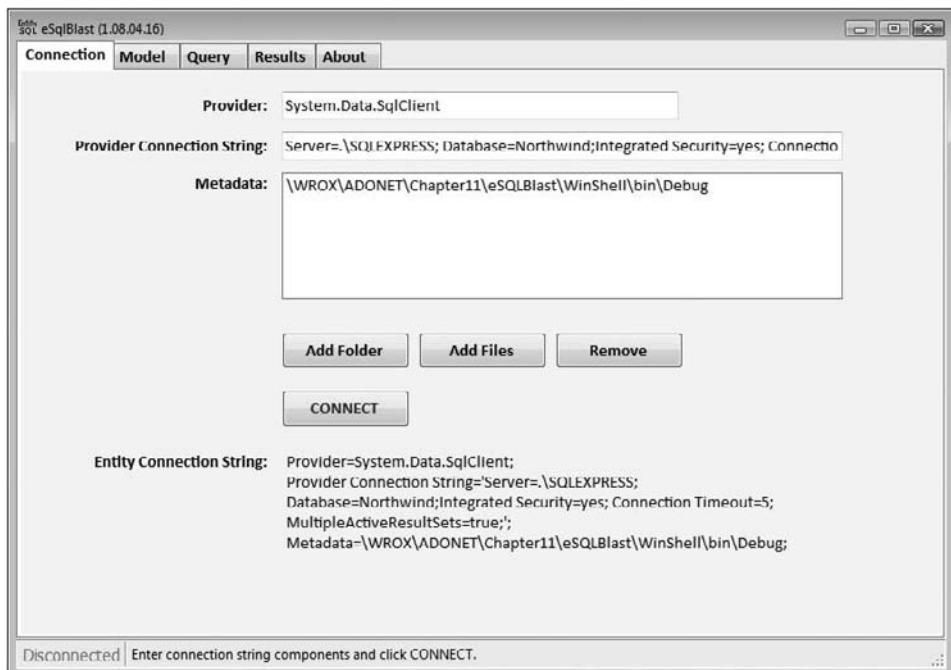


Figure 11-1

Change the default values, if necessary, and click the Connect button to open a connection, which enables the model and query operations.

*Northwind EntityType names have been singularized; EntitySet names are plural. Change the default connection string if you're not using a local instance of SQL Server 2005+ Express.*

## Model Page

The Model page requires the well-formed path to the `ModelName.csdl` file, `Northwind.csdl` for this example. Clicking the Execute button performs an XSLT transform on the file to create an HTML page with individual tables for EntityContainers, EntityTypes, ComplexTypes, and Associations (see Figure 11-2). The transform file is 290 lines and has been modified to substitute the 11-point Calibri for the 10-point Verdana font and reduce vertical whitespace.

## Part IV: Introducing the ADO.NET Entity Framework

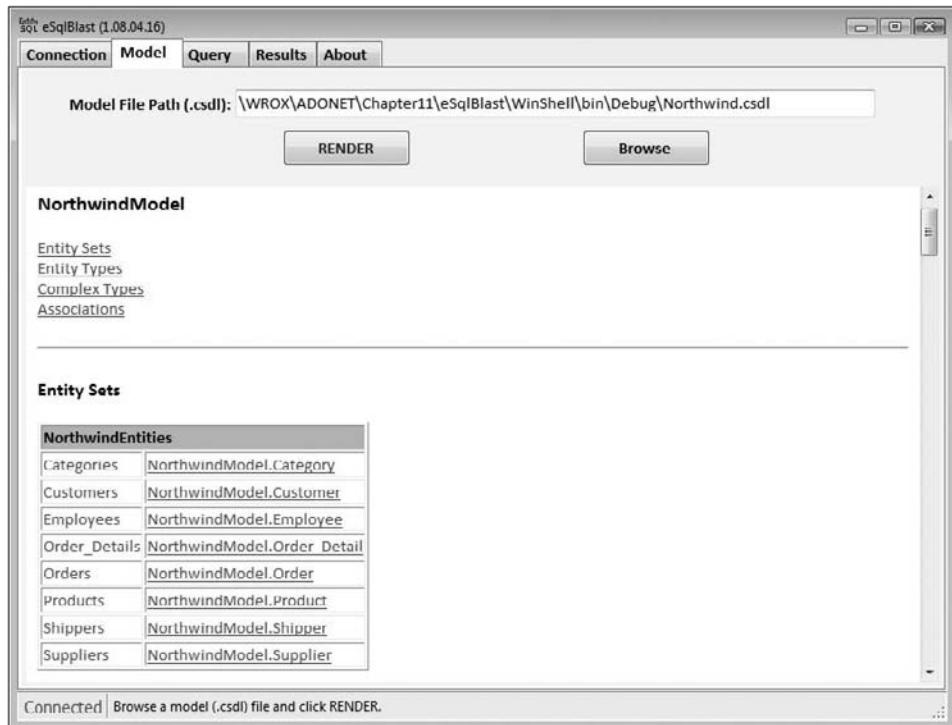


Figure 11-2

Clicking an anchor exposes the target table. EntityType and Association names have been singularized.

## Query Page

The Query page contains a text box in which to type eSQL queries. Keywords are color-coded and popup lists for IntelliSense let you select the entity model and entity set, and then press Enter to insert the name in the query text (see Figure 11-3).

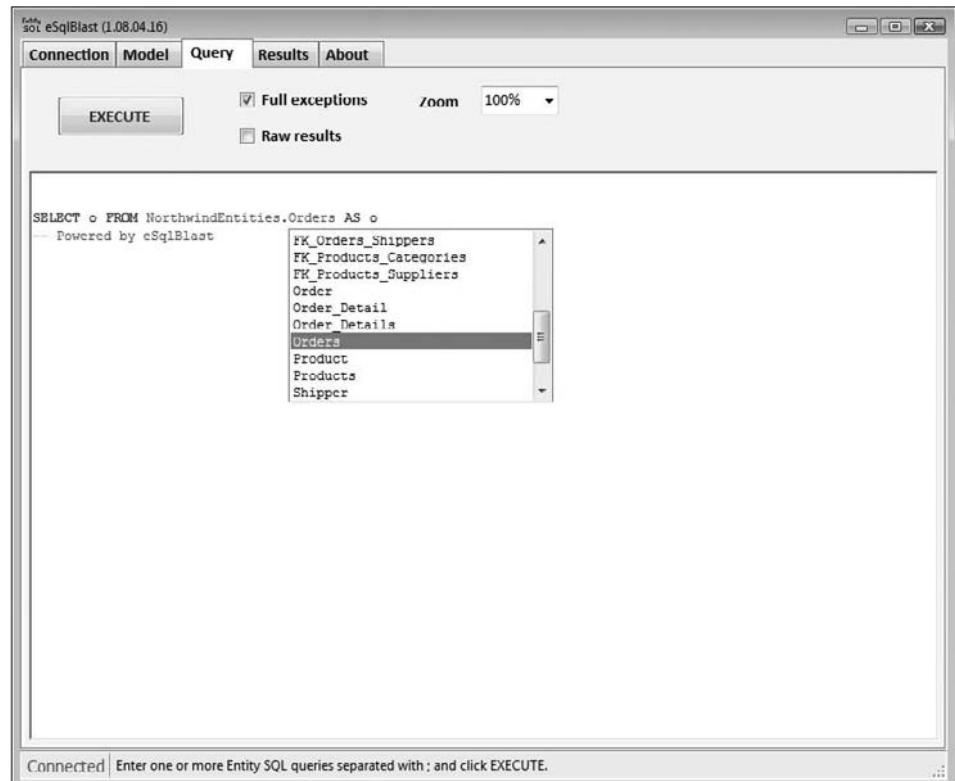


Figure 11-3

The simple eSQL query for this example is `SELECT o FROM NorthwindEntities.Orders AS o`. Marking the Full Exceptions check box returns the `Exception.InnerException` text instead of the `Exception.Message` text to the Result page.

## Result Page

Clicking Execute on the Query page opens the Result page with the eSQL query, store query in the RDBMS's native query language, and record count (see Figure 11-4).

## Part IV: Introducing the ADO.NET Entity Framework

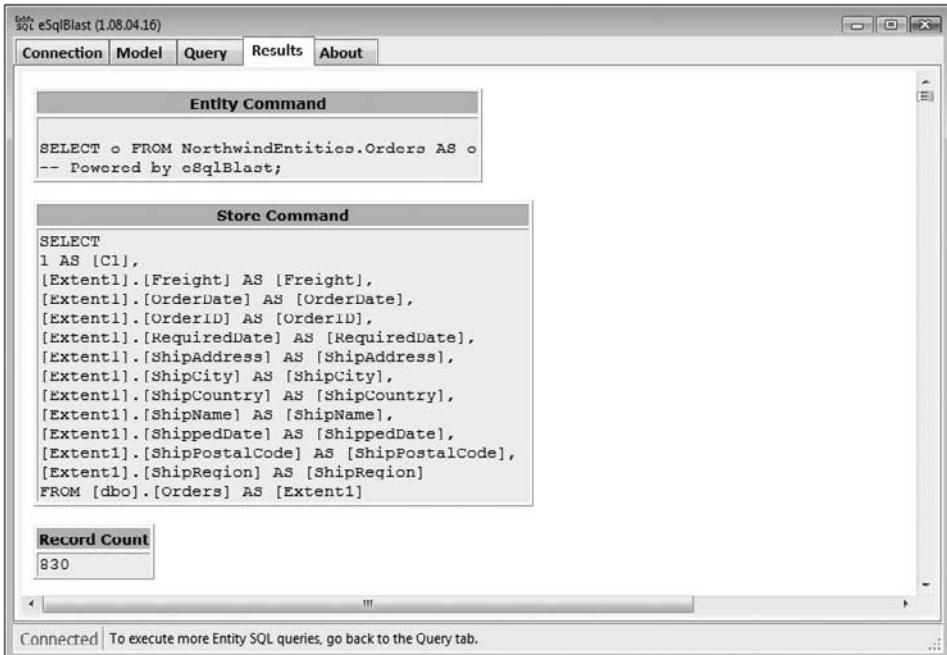


Figure 11-4

Below the record count is an individual table for each member of the entity set, if the VALUE modifier isn't used in the SELECT statement (see Figure 11-5).

Freight	OrderDate	OrderID	RequiredDate	ShipAddress	ShipCity	ShipCountry	ShipName	ShippedDat
32.3800	12/00:00 AM	10248	8/1/1996 12:00:00 AM	59 rue de l'Abbaye	Reims	France	Vins et alcools Chevalier	7/16/1996 12:00:00 AM
11.6100	12/00:00 AM	10249	8/16/1996 12:00:00 AM	Luisenstr. 48	Munster	Germany	Toms Spezialitäten	7/10/1996 12:00:00 AM
65.8300	12/00:00 AM	10250	8/5/1996 12:00:00 AM	Rua do Paço, 67	Rio de Janeiro	Brazil	Hanari Carnes	7/12/1996 12:00:00 AM
41.3400	12/00:00 AM	10251	8/5/1996 12:00:00 AM	2, rue du Commerce	Lyon	France	Victuailles en stock	7/15/1996 12:00:00 AM
51.3000	12/00:00 AM	10252	8/6/1996 12:00:00 AM	Boulevard Tirou, 255	Charleroi	Belgium	Suprèmes délices	7/11/1996 12:00:00 AM
50.1700	12/00:00 AM	10253	7/24/1996 12:00:00 AM	Rua do Paço, 67	Rio de Janeiro	Brazil	Hanari Carnes	7/16/1996 12:00:00 AM

Figure 11-5

Marking the Query page's Raw Results check box returns the resultset as an XML document (see Figure 11-6).



The screenshot shows the eSQLBlast application window. The title bar reads "Edit: sql eSQLBlast (1.08.04.16)". The menu bar includes "File", "Edit", "Connection", "Model", "Query", "Results", and "About". The "Results" tab is selected. The main pane displays the following Entity SQL query results as XML:

```

<?xml version="1.0" encoding="utf-8" ?>
- <Batch>
- <Command>
  <EntityStatement>SELECT o FROM NorthwindEntities.Orders AS o -- Powered by
    eSQLBlast;</EntityStatement>
<StoreStatement>SELECT 1 AS [C1], [Extent1].[Freight] AS [Freight], [Extent1].[OrderDate] AS
  [OrderDate], [Extent1].[OrderID] AS [OrderID], [Extent1].[RequiredDate] AS
  [RequiredDate], [Extent1].[ShipAddress] AS [ShipAddress], [Extent1].[ShipCity] AS
  [ShipCity], [Extent1].[ShipCountry] AS [ShipCountry], [Extent1].[ShipName] AS
  [ShipName], [Extent1].[ShippedDate] AS [ShippedDate], [Extent1].[ShipPostalCode] AS
  [ShipPostalCode], [Extent1].[ShipRegion] AS [ShipRegion] FROM [dbo].[Orders] AS
  [Extent1]</StoreStatement>
- <Collection>
- <Record>
- <o>
  - <Entity>
    <Freight>32.3800</Freight>
    <OrderDate>7/4/1996 12:00:00 AM</OrderDate>
    <OrderID>10248</OrderID>
    <RequiredDate>8/1/1996 12:00:00 AM</RequiredDate>
    <ShipAddress>59 rue de l'Abbaye</ShipAddress>
    <ShipCity>Reims</ShipCity>
    <ShipCountry>France</ShipCountry>
    <ShipName>Vins et alcools Chevalier</ShipName>
    <ShippedDate>7/16/1996 12:00:00 AM</ShippedDate>
    <ShipPostalCode>51100</ShipPostalCode>
    <ShipRegion Null="True" />
  </Entity>
</o>
</Record>

```

At the bottom left, it says "Connected | To execute more Entity SQL queries, go back to the Query tab."

Figure 11-6

## Understanding How Entity SQL Differs from Transact-SQL

The “Writing EntityQueries in Entity SQL” topic of Chapter 9 provided an abbreviated list of the differences between eSQL and T-SQL. The sections that follow show you where eSQL syntax departs significantly from that of T-SQL.

eSQL provides a set of canonical functions which the Entity Framework expects all entity-enabled data providers to support. These functions, which are a subset of T-SQL functions as supported by the SqlClient managed provider, fall into the following categories:

- Aggregate canonical functions:** AVG(), BIGCOUNT(), COUNT(), MIN(), MAX(), SUM()
- Math canonical functions:** ABS(), CEILING(), FLOOR(), ROUND()
- String canonical functions:** CONCAT(), INDEXOF(), LEFT(), LENGTH(), LTRIM(), RIGHT(), TRIM(), REPLACE(), REVERSE(), RTRIM(), SUBSTRING(), TOLOWER(), TOUPPER()

## Part IV: Introducing the ADO.NET Entity Framework

---

- ❑ **Date and time canonical functions:** DATEADD(), DATEDIFF(), DAY(), GETDATE(), GETUTCDATE(), HOUR(), MINUTE(), MONTH(), SECOND(), YEAR()
- ❑ **Bitwise canonical functions:** BitWiseAnd(), BitWiseNot(), BitWiseOr(), BitWiseXor()
- ❑ **Other canonical functions:** NewGuid()

eSQL supports the following `SqlClient` functions that aren't included in the basic set of canonical functions:

- ❑ **Aggregate functions:** CHECKSUM\_AGG(), STDDEV(), STDDEVP(), VAR(), VARP()
- ❑ **Math functions:** ACOS(), ASCII(), ASIN(), ATAN(), ATN2(), COS(), DEGREES(), EXP(), LOG(), LOG10(), RADIANS(), RAND(), SIGN(), SIN(), SQRT(), SQUARE(), PI(), POWER()
- ❑ **String functions:** CHARINDEX(), DIFFERENCE(), LEN(), LIKE(), LOWER(), LTRIM(), PATINDEX(), REPLACE(), REPLICATE(), REVERSE(), RTRIM(), SOUNDEX(), SPACE(), STUFF(), UPPER()
- ❑ **Date and time functions:** CURRENT\_TIMESTAMP(), DATENAME(), DATEPART()
- ❑ **System functions:** CHECKSUM(), CURRENT\_TIMESTAMP(), CURRENT\_USER(), DATALENGTH(), HOST\_NAME(), ISDATE(), ISNUMERIC(), NEWID(), USER\_NAME()

eSQL also supports the following operators in common with T-SQL and the `SqlClient` provider:

- ❑ **Arithmetic operators:** +, /, %, \*, - (Negative), - (Subtract)
- ❑ **Comparison operators:** [NOT] BETWEEN, =, >, >=, IS [NOT] NULL, <, <=, [NOT] LIKE, !=, <>
- ❑ **Logical operators:** AND (&&), NOT (!), OR (||)
- ❑ **Other operators:** . (Member Access), + (String Concatenation), CAST (Type)

The following expressions are shared by T-SQL and eSQL:

- ❑ **Select expressions:** SELECT, TOP (n), WHERE
- ❑ **Case expressions:** CASE, ELSE, THEN, WHEN

The following sections describe the most important specific differences between eSQL and T-SQL.

*The NorthwindEntitySqlCS.sln sample project, which the later “Measuring the Performance Penalty of EntitySQL Queries” section describes, generates the T-SQL versions of sample eSQL queries by invoking the `EntityCommand.ToTraceString()` method. The same method provides the text for the unformatted `<StoreStatement>` element of the eSQLBlast project. The T-SQL statements that the canonical query tree generates are quite complex and often not what you would expect.*

## **Entity Alias Prefixes Are Mandatory**

References to entities and entity properties in `SELECT` projections, as well as `GROUP BY` and `ORDER BY` clauses must use an entity alias assigned by the `FROM Container.EntitySet AS Alias` clause, as in the following example:

## eSQL

```
SELECT o FROM NorthwindEntities.Orders AS o;
```

which returns the Northwind Orders EntitySet as a collection of Order entities in nested DbDataRecords that implement `IExtendedDataRecord`. This is the simplest eSQL query you can execute against a data store. eSQLBlast's Results pane displays each `SqlDataRecord` in an individual table.

Here's the corresponding T-SQL statement sent to the Northwind data store:

## T-SQL

```
SELECT
    1 AS [C1],
    [Extent1].[OrderID] AS [OrderID],
    [Extent1].[OrderDate] AS [OrderDate],
    [Extent1].[RequiredDate] AS [RequiredDate],
    [Extent1].[ShippedDate] AS [ShippedDate],
    [Extent1].[Freight] AS [Freight],
    [Extent1].[ShipName] AS [ShipName],
    [Extent1].[ShipAddress] AS [ShipAddress],
    [Extent1].[ShipCity] AS [ShipCity],
    [Extent1].[ShipRegion] AS [ShipRegion],
    [Extent1].[ShipPostalCode] AS [ShipPostalCode],
    [Extent1].[ShipCountry] AS [ShipCountry]
FROM [dbo].[Orders] AS [Extent1]
```

## Explicit Projections Are Required

eSQL doesn't support the `*` all-columns wildcard for projections. To return a projection, you must use

`SELECT EntityAlias1.PropertyName1, EntityAlias2.PropertyName2, ... FROM Container.`

`EntitySet AS Alias` as in:

## eSQL

```
SELECT o.OrderID, o.OrderDate, o.RequiredDate, o.ShippedDate, o.Freight, ...
      o.ShipCountry FROM NorthwindEntities.Orders AS o;
```

The T-SQL version is the same as that in the preceding section.

## The **VALUE** Modifier Flattens Results

`SELECT VALUE Alias FROM Container.EntitySet AS o` returns a flattened entity as an unnested `DbDataRecord`. `SELECT VALUE` can only project a single item. Therefore:

## eSQL

```
SELECT VALUE o.ShippedDate FROM NorthwindEntities.Orders AS o;
```

## Part IV: Introducing the ADO.NET Entity Framework

is valid and returns a single scalar `DateTime` column but

### eSQL

```
SELECT VALUE o.OrderID, o.ShippedDate FROM NorthwindEntities.Orders AS o;
```

throws a “`SELECT VALUE` can have only one expression in the projection list, near . . .” exception.

## Dot-Notation Syntax Returns Many:One Navigation Properties

`EntityAlias.NavigationEntity.NavigationProperty` syntax can return missing foreign-key values or replace foreign-key values with readable names when `NavigationEntity` represents the 1 side of a many:one relationship. For example the following eSQL statement

### eSQL

```
SELECT TOP(5) o.OrderID, o.Customer.CustomerID AS CustID,
       o.Employee.EmployeeID AS EmplID, o.OrderDate AS [Date],
       o.Shipper.ShipperID AS ShipID, o.Order_Details
  FROM NorthwindEntities.Orders AS o;
```

returns a nested `DbDataReader` containing properties from an `Order` entity, and one property each from associated `Customer`, `Employee`, and `Shipper` entities in a `DbDataReader` that contains an associated `Order_Detail` collection for each row, as shown in Figure 11-7.

eSQLBlast (1.08.04.16)									
Connection	Model	Query	Results	About					
OrderID	CUSTID	EmplID	Date	ShipID	Order_Details				
10248	VINET	5	7/4/1996 12:00:00 AM	3	Discount	OrderID	ProductID	Quantity	UnitPrice
					0	10248	11	12	14.0000
					0	10248	42	10	9.8000
				1	0	10248	72	5	34.8000
10249	TOMSP	6	7/5/1996 12:00:00 AM		Discount	OrderID	ProductID	Quantity	UnitPrice
					0	10249	14	9	18.6000
					0	10249	51	40	42.4000
10250	HANAR	4	7/8/1996 12:00:00 AM	2	Discount	OrderID	ProductID	Quantity	UnitPrice
					0	10250	41	10	7.0000
					0.15	10250	51	35	42.4000
				1	0.15	10250	65	15	16.8000
10251	VICTE	3	7/8/1996 12:00:00 AM		Discount	OrderID	ProductID	Quantity	UnitPrice
					0.05	10251	22	6	16.8000
					0.05	10251	57	15	15.6000
				2	0	10251	65	20	16.0000
10252	SUPRD	4	7/9/1996 12:00:00 AM		Discount	OrderID	ProductID	Quantity	UnitPrice
					0.05	10252	20	40	64.8000
					0.05	10252	33	25	2.0000
					0	10252	60	40	27.2000

Connected To execute more Entity SQL queries, go back to the Query tab.

Figure 11-7

Here's the T-SQL equivalent:

## T-SQL

```
SELECT
    [Project2].[OrderID] AS [OrderID],
    [Project2].[OrderDate] AS [OrderDate],
    [Project2].[CustomerID] AS [CustomerID],
    [Project2].[EmployeeID] AS [EmployeeID],
    [Project2].[ShipperID] AS [ShipperID],
    [Project2].[C1] AS [C1],
    [Project2].[C2] AS [C2],
    [Project2].[OrderID1] AS [OrderID1],
    [Project2].[ProductID] AS [ProductID],
    [Project2].[UnitPrice] AS [UnitPrice],
    [Project2].[Quantity] AS [Quantity],
    [Project2].[Discount] AS [Discount]
FROM (
    SELECT
        [Limit1].[OrderID] AS [OrderID],
        [Limit1].[OrderDate] AS [OrderDate],
        [Limit1].[CustomerID] AS [CustomerID],
        [Limit1].[EmployeeID] AS [EmployeeID],
        [Limit1].[ShipperID] AS [ShipperID],
        [Limit1].[C1] AS [C1],
        [Extent5].[OrderID] AS [OrderID1],
        [Extent5].[ProductID] AS [ProductID],
        [Extent5].[UnitPrice] AS [UnitPrice],
        [Extent5].[Quantity] AS [Quantity],
        [Extent5].[Discount] AS [Discount],
        CASE WHEN ([Extent5].[OrderID] IS NULL)
            THEN CAST(NULL AS int) ELSE 1 END AS [C2]
    FROM (SELECT TOP (5)
        [Extent1].[OrderID] AS [OrderID],
        [Extent1].[OrderDate] AS [OrderDate],
        [Extent2].[CustomerID] AS [CustomerID],
        [Extent3].[EmployeeID] AS [EmployeeID],
        [Extent4].[ShipperID] AS [ShipperID],
        1 AS [C1]
    FROM [dbo].[Orders] AS [Extent1]
    LEFT OUTER JOIN [dbo].[Customers] AS [Extent2]
        ON [Extent1].[CustomerID] = [Extent2].[CustomerID]
    LEFT OUTER JOIN [dbo].[Employees] AS [Extent3]
        ON [Extent1].[EmployeeID] = [Extent3].[EmployeeID]
    LEFT OUTER JOIN [dbo].[Shippers] AS [Extent4]
        ON [Extent1].[ShipVia] = [Extent4].[ShipperID] ) AS [Limit1]
    LEFT OUTER JOIN [dbo].[Order Details] AS [Extent5]
        ON [Limit1].[OrderID] = [Extent5].[OrderID]
) AS [Project2]
ORDER BY [Project2].[OrderID] ASC, [Project2].[CustomerID] ASC,
[Project2].[EmployeeID] ASC, [Project2].[ShipperID] ASC, [Project2].[C2] ASC;
```

*The Canonical Query Tree (CQT) adds the ORDER BY clause as a result of applying the TOP(5) operator.*

## Part IV: Introducing the ADO.NET Entity Framework

## **Nested Queries Are Required for One:Many Navigation Properties**

The eSQL query engine throws a run-time exception if you attempt to use dot-notation syntax to return navigation properties of navigation entities having a one:many relationship with a parent entity. To access these properties, you must use a nested SELECT query whose data source is the navigation entity, `o.Order_Details` in the following eSQL command:

eSQL

```
SELECT TOP(5) o.OrderID, o.Customer.CustomerID AS CustID,
    o.Employee.EmployeeID AS EmplID, o.OrderDate AS [Date],
    o.Shipper.ShipperID AS ShipID,
    (SELECT d.Quantity AS [Quan.], d.Product.Category.CategoryName AS [Category],
        d.ProductID AS SKU, d.Product.ProductName, d.UnitPrice AS Price,
        d.Discount AS [Disc.]
    FROM o.Order_Details AS d)
FROM NorthwindEntities.Orders AS o;
```

The preceding query's subquery includes references from Product and Category many:one navigation entities (see Figure 11-8).

Northwind Entity SQL Query Results										
Connection		Model		Query		Results				
OrderID	CustID	EmplID	Date	ShipID	x0023_x0023_0					
10248	VINET	5	7/4/1996 12:00:00 AM	3	Quan.	Category	SKU	ProductName	Price	Disc.
					12	Dairy Products	11	Queso Cabrales	14.0000	0
					10	Grains/Cereals	42	Singaporean Hokkien Fried Mee	9.8000	0
10249	TOMSP	6	7/5/1996 12:00:00 AM	1	Quan.	Category	SKU	ProductName	Price	Disc.
					9	Produce	14	Tofu	18.6000	0
					40	Produce	51	Manjimup Dried Apples	42.4000	0
10250	HANAR	4	7/8/1996 12:00:00 AM	2	Quan.	Category	SKU	ProductName	Price	Disc.
					10	Seafood	41	Jack's New England Clam Chowder	7.7000	0
					35	Produce	51	Manjimup Dried Apples	42.4000	0.15
10251	VICTE	3	7/8/1996 12:00:00 AM	1	Quan.	Category	SKU	ProductName	Price	Disc.
					6	Grains/Cereals	22	Gustaf's Knäckebröd	16.8000	0.05
					15	Grains/Cereals	57	Ravioli Ariosto	15.6000	0.05
Connected		To execute more Entity SQL queries, go back to the Query tab.								

**Figure 11-8**

You can use a nested SELECT query, which must be enclosed within parenthesis, anywhere in an eSQL query where a value of the nested query's type would be valid. In the preceding example, the subquery type is part of a projection.

The T-SQL version of this eSQL query is quite similar to the preceding T-SQL example except for the elements that return the Order\_Details projection.

## **JOINS Are the Last Resort**

JOINS are supported primarily for combining entity instances without navigation properties. JOINS are less efficient than references to an associated entity or EntitySet because they flatten the data model. If navigation properties are present, use them.

## **NAVIGATE Is a Complex Substitute for Dot Notation or Nested Queries**

The NAVIGATE keyword provides an alternative — but cumbersome — method for traversing associations. Here's the generic syntax for the NAVIGATE operator:

```
NAVIGATE(instance-expression, relationship-type, [to-end [, from-end]]
```

where *instance-expression* is an instance of an EntityType and *relationship-type* is a namespace-prefixed AssociationSet, *ModelName*.AssociationSet. You only need to provide *to-end* and *from-end* identifiers if name resolution in the *relationship-type* is ambiguous.

*Many examples in eSQL documentation that use the NAVIGATE keyword either omit the ModelName namespace prefix or use the same value (Northwind) for ModelName and Container. Omitting the ModelName throws an exception when attempting to execute the query. Unless the ModelName and Container names are identical, exchanging the latter for the former also throws an error.*

The following NAVIGATE version of the earlier “Dot-Notation Syntax Returns Many:One Navigation Properties” section’s dot-notation example also returns the results shown in Figure 11-7 with a similar T-SQL statement:

### **eSQL**

```
SELECT TOP(5) o.OrderID,
        DEREF(NAVIGATE(o, NorthwindModel.FK_Orders_Customers)).CustomerID AS CustID,
        DEREF(NAVIGATE(o, NorthwindModel.FK_Orders_Employees)).EmployeeID AS EmplID,
        o.OrderDate AS [Date],
        DEREF(NAVIGATE(o, NorthwindModel.FK_Orders_Shippers)).ShipperID AS ShipID,
        (SELECT VALUE DEREF(d)
         FROM NAVIGATE(o, NorthwindModel.FK_Order_Details_Orders) AS d) AS Order_Details
        FROM NorthwindEntities.Orders AS o;
```

## Part IV: Introducing the ADO.NET Entity Framework

---

NAVIGATE returns references to entities (`EntityReferences`), not entity instances, because AssociationSets are collections of individual references or reference collections. The DREF() operator returns a Customer, Employee, or Shipper EntityType instance to deliver its primary key value to the projection; the Customers, Employees or Shippers EntitySet is the to-end and Orders is the from-end for these many:one relationships.

*The reason for inverting the normal sequence of from-end, to-end arguments isn't clear.*

The DREF(d) operator returns associated Order\_Details instances from a one:many association.

## ***REF, DEREF, CREATEREF, ROW, and KEY Manage Entity References***

`REF(EntityType)` creates a reference from an entity. As an example, executing:

### eSQL

```
SELECT TOP(5) REF(o) FROM NorthwindEntities.Orders AS o;
```

### T-SQL

```
SELECT
[Limit1].[C1] AS [C1],
[Limit1].[OrderID] AS [OrderID]
FROM ( SELECT TOP (5)
[Extent1].[OrderID] AS [OrderID],
1 AS [C1]
FROM [dbo].[Orders] AS [Extent1]
) AS [Limit1]
```

creates five references, which appear as five single-row tables with two columns, EntitySet and OrderID, in eSQLBlast's Results page. To return a property value, add the property name, as in:

### eSQL

```
SELECT TOP(5) REF(o).OrderDate FROM NorthwindEntities.Orders AS o;
```

### T-SQL

```
SELECT
[Limit1].[C1] AS [C1],
[Limit1].[OrderDate] AS [OrderDate]
FROM ( SELECT TOP (5)
[Extent1].[OrderDate] AS [OrderDate],
1 AS [C1]
FROM [dbo].[Orders] AS [Extent1]
) AS [Limit1]
```

which returns a five-row table with an OrderDate column. A REF (*n*) to a nonexistent entity *n* returns NULL.

*The terms reference and association aren't synonyms. An Association is a named reference defined in the conceptual layer by the Name.csdl file and a members of an EntityType's AssociationSet.*

DEREF(*es*[.expression]) FROM *EntityType* AS *es* creates an entity instance from a reference, as in:

## eSQL

```
SELECT TOP(5) DEREF(REF(o)) FROM NorthwindEntities.Orders AS o;
```

## T-SQL

```
SELECT  
[Limit1].[C1] AS [C1],  
[Limit1].[OrderID] AS [OrderID],  
[Limit1].[OrderDate] AS [OrderDate],  
[Limit1].[RequiredDate] AS [RequiredDate],  
[Limit1].[ShippedDate] AS [ShippedDate],  
[Limit1].[Freight] AS [Freight],  
[Limit1].[ShipName] AS [ShipName],  
[Limit1].[ShipAddress] AS [ShipAddress],  
[Limit1].[ShipCity] AS [ShipCity],  
[Limit1].[ShipRegion] AS [ShipRegion],  
[Limit1].[ShipPostalCode] AS [ShipPostalCode],  
[Limit1].[ShipCountry] AS [ShipCountry]  
FROM ( SELECT TOP (5)  
[Extent1].[OrderID] AS [OrderID],  
[Extent1].[OrderDate] AS [OrderDate],  
[Extent1].[RequiredDate] AS [RequiredDate],  
[Extent1].[ShippedDate] AS [ShippedDate],  
[Extent1].[Freight] AS [Freight],  
[Extent1].[ShipName] AS [ShipName],  
[Extent1].[ShipAddress] AS [ShipAddress],  
[Extent1].[ShipCity] AS [ShipCity],  
[Extent1].[ShipRegion] AS [ShipRegion],  
[Extent1].[ShipPostalCode] AS [ShipPostalCode],  
[Extent1].[ShipCountry] AS [ShipCountry],  
1 AS [C1]  
FROM [dbo].[Orders] AS [Extent1]  
) AS [Limit1]
```

which creates five 11-column tables, or:

## eSQL

```
SELECT VALUE TOP(5) DEREF(REF(o)) FROM NorthwindEntities.Orders AS o;
```

## Part IV: Introducing the ADO.NET Entity Framework

---

### T-SQL

```
SELECT TOP (5)
[c].[OrderID] AS [OrderID],
[c].[OrderDate] AS [OrderDate],
[c].[RequiredDate] AS [RequiredDate],
[c].[ShippedDate] AS [ShippedDate],
[c].[Freight] AS [Freight],
[c].[ShipName] AS [ShipName],
[c].[ShipAddress] AS [ShipAddress],
[c].[ShipCity] AS [ShipCity],
[c].[ShipRegion] AS [ShipRegion],
[c].[ShipPostalCode] AS [ShipPostalCode],
[c].[ShipCountry] AS [ShipCountry]
FROM [dbo].[Orders] AS [c]
```

which generates a five-row, 11-column table. A DEREF(*n*) to a nonexistent reference *n* returns NULL.

CREATEREF(*entityset\_identifier*, *record\_typed\_expression*) creates a single reference from an EntitySet and an entity's key properties. The *record\_typed\_expression* argument specifies the number of entity properties so the result is structurally equivalent to the entity's key\_type. For example, the following expression creates a reference to the first Order\_Detail instance of Order 10248:

### eSQL

```
CREATEREF(NorthwindEntities.Order_Details, ROW(10248, 11));
```

### T-SQL

```
SELECT
10248 AS [C1],
11 AS [C2]
FROM (SELECT cast(1 as bit) AS X ) AS [SingleRowTable1]
```

and generates a single-row table with EntitySet, OrderID and ProductID columns.

KEY(*createref\_expression*) is the inverse of CREATEREF(). This expression:

### eSQL

```
SELECT TOP(5) KEY(CREATEREF(NorthwindEntities.Order_Details, ROW(10248, 11)))
FROM NorthwindEntities.Order_Details AS o;
```

creates a five-row table with only the key value columns, OrderID and ProductID.

### T-SQL

```
SELECT
[Limit1].[C1] AS [C1],
[Limit1].[C2] AS [C2],
[Limit1].[C3] AS [C3],
[Limit1].[C4] AS [C4]
FROM ( SELECT TOP (5)
1 AS [C1],
```

```
1 AS [C2],  
10248 AS [C3],  
11 AS [C4]  
FROM [dbo].[Order Details] AS [Extent1]  
) AS [Limit1]
```

## Type Constructors Create ROWs, Multisets, and Instances of EntityTypes

`ROW(expression [AS alias] [,...])` is a type constructor operator that creates an anonymous `DbDataRecord`, as demonstrated in the preceding section.

`MULTISET(expression [, expression ...])` is a type constructor operator that creates an anonymous, hierarchical `DbDataRecord` to represents a collection. You can create a `MULTISET` collection of integers, such as `OrderID` values, with this code:

### eSQL

```
MULTISET(10282, 10292, 10302, 10312, 10322);
```

or the shorthand version:

### eSQL

```
{10282, 10292, 10302, 10312, 10322};
```

and then use it in a query such as the following:

### eSQL

```
SELECT VALUE o FROM NorthwindEntities.Orders AS o  
WHERE o.OrderID IN MULTISET(10282, 10292, 10302, 10312, 10322);
```

to return a flattened set of five `Order` instances.

### T-SQL

```
ELECT  
[Extent1].[OrderID] AS [OrderID],  
[Extent1].[OrderDate] AS [OrderDate],  
[Extent1].[RequiredDate] AS [RequiredDate],  
[Extent1].[ShippedDate] AS [ShippedDate],  
[Extent1].[Freight] AS [Freight],  
[Extent1].[ShipName] AS [ShipName],  
[Extent1].[ShipAddress] AS [ShipAddress],  
[Extent1].[ShipCity] AS [ShipCity],  
[Extent1].[ShipRegion] AS [ShipRegion],  
[Extent1].[ShipPostalCode] AS [ShipPostalCode],  
[Extent1].[ShipCountry] AS [ShipCountry]  
FROM [dbo].[Orders] AS [Extent1]  
WHERE ([Extent1].[OrderID] = 10282) OR ([Extent1].[OrderID] = 10292) OR  
([Extent1].[OrderID] = 10302) OR ([Extent1].[OrderID] = 10312) OR  
([Extent1].[OrderID] = 10322)
```

## Part IV: Introducing the ADO.NET Entity Framework

---

*It's surprising that the Canonical Query Tree doesn't construct the simpler WHERE [Extent1].[OrderID] IN (10282, 10292, 10302, 10312, 10322) clause.*

The Named Type Constructor creates instances of the specified type by supplying values of its properties. For example, the following creates an instance of Order 10282 with missing Address, ShipAddress, ShipRegion, and ShipPostalCode property values:

### eSQL

```
SELECT VALUE NorthwindModel.Order(o.OrderID, o.OrderDate, o.RequiredDate, null,
    o.Freight, o.ShipName, null, o.ShipCity, null, null, o.ShipCountry)
FROM NorthwindEntities.Orders AS o
WHERE o.OrderID = 10282;
```

The projection must have the same number of type-compatible entries as the entity type.

### T-SQL

```
SELECT
    CAST(NULL AS int) AS [C1],
    [Extent1].[OrderID] AS [OrderID],
    [Extent1].[OrderDate] AS [OrderDate],
    [Extent1].[RequiredDate] AS [RequiredDate],
    CAST(NULL AS datetime) AS [C2],
    [Extent1].[Freight] AS [Freight],
    [Extent1].[ShipName] AS [ShipName],
    CAST(NULL AS nvarchar(60)) AS [C3],
    [Extent1].[ShipCity] AS [ShipCity],
    CAST(NULL AS nvarchar(15)) AS [C4],
    CAST(NULL AS nvarchar(10)) AS [C5],
    [Extent1].[ShipCountry] AS [ShipCountry]
FROM [dbo].[Orders] AS [Extent1]
WHERE [Extent1].[OrderID] = 10282
```

## ***The UNION, INTERSECT, OVERLAPS, and EXCEPT Set Operators Require Sub-Queries***

Unlike T-SQL, the *(select-expression) set-operator (select-expression)* syntax requires *select-expression* to be enclosed within parentheses, as in:

### eSQL

```
(SELECT c.CompanyName, c.City, c.Country, 'Customer' AS Category
    FROM NorthwindEntities.Customers AS c)
UNION
(SELECT s.CompanyName, s.City, s.Country, 'Supplier' AS Category
    FROM NorthwindEntities.Suppliers AS s) AS u
```

OVERLAPS determines whether two collections have a common value; *(expression) OVERLAPS (expression)* is the equivalent of *(expression) OVERLAPS (expression)*.

## **Sorting Collections Returned by Set Operators Requires a Nested Query**

T-SQL enables applying the GROUP BY operator following the second select-expression of a set operator, as in:

### **T-SQL**

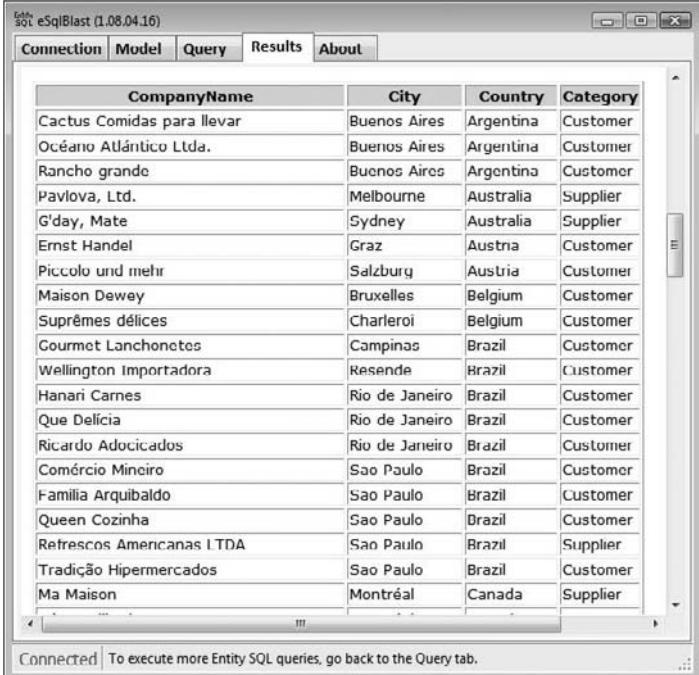
```
SELECT CompanyName, City, Country, 'Customer' AS Category FROM Customers
UNION
SELECT CompanyName, City, Country, 'Supplier' AS Category FROM Suppliers
ORDER BY Country, City;
```

eSQL requires nesting the UNION query structure in an outer SELECT expression, which is emphasized in this example:

### **eSQL**

```
SELECT u.CompanyName, u.City, u.Country, u.Category FROM
    (SELECT c.CompanyName, c.City, c.Country, 'Customer' AS Category
     FROM NorthwindEntities.Customers AS c)
UNION
    (SELECT s.CompanyName, s.City, s.Country, 'Supplier' AS Category
     FROM NorthwindEntities.Suppliers AS s) AS u
ORDER BY u.Country, u.City;
```

The preceding query returns the result shown in Figure 11-9.



The screenshot shows the eSqlBlast application interface. The title bar reads "eSqlBlast (1.08.04.16)". Below the title bar is a menu bar with tabs: Connection, Model, Query, Results, and About. The "Results" tab is currently selected. The main area displays a table with four columns: CompanyName, City, Country, and Category. The data in the table is as follows:

CompanyName	City	Country	Category
Cactus Comidas para llevar	Buenos Aires	Argentina	Customer
Océano Atlántico Ltda.	Buenos Aires	Argentina	Customer
Rancho grande	Buenos Aires	Argentina	Customer
Pavlova, Ltd.	Melbourne	Australia	Supplier
G'day, Mate	Sydney	Australia	Supplier
Ernst Handel	Graz	Austria	Customer
Piccolo und mehr	Salzburg	Austria	Customer
Maison Dewey	Bruxelles	Belgium	Customer
Suprêmes délices	Charleroi	Belgium	Customer
Gourmet Lanchonetes	Campinas	Brazil	Customer
Wellington Importadora	Resende	Brazil	Customer
Hanari Carnes	Rio de Janeiro	Brazil	Customer
Que Delícia	Rio de Janeiro	Brazil	Customer
Ricardo Adocicados	Rio de Janeiro	Brazil	Customer
Comércio Mineiro	Sao Paulo	Brazil	Customer
Família Arquibaldo	Sao Paulo	Brazil	Customer
Queen Cozinha	Sao Paulo	Brazil	Customer
Refrescos Americanas LTDA	Sao Paulo	Brazil	Supplier
Tradição Hipermercados	Sao Paulo	Brazil	Customer
Ma Maison	Montréal	Canada	Supplier

At the bottom of the application window, there is a status bar with the text "Connected | To execute more Entity SQL queries, go back to the Query tab."

Figure 11-9

### **Set Operators *ANYELEMENT* and *FLATTEN***

#### **Work on Collections**

*ANYELEMENT(collection)* returns the or an arbitrary element if collection has one or more elements, and *NULL* otherwise.

*FLATTEN(collection\_of\_collections)* turns a nested *collection\_of\_collections* into a flattened collection without a nesting. *FLATTEN* performs the same operation as `SELECT VALUE`.

### **SKIP and LIMIT Sub-Clauses of the ORDER BY Clause**

#### **Handle Paging**

Although eSQL accepts T-SQL's `TOP(n)` modifier without an `ORDER BY` clause, the preferred approach for paging is to combine the `SKIP(n)` and `LIMIT(n)` sub-clauses with the mandatory `ORDER BY` clause, as in this example:

#### **eSQL**

```
SELECT o.OrderID, o.Customer.CustomerID AS CustID,
       o.Employee.EmployeeID AS EmplID, o.OrderDate AS [Date],
       o.Shipper.ShipperID AS ShipID, o.Order_Details
  FROM NorthwindEntities.Orders AS o
 ORDER BY o.OrderID DESC
 SKIP(10)
 LIMIT(5);
```

which returns Orders 11063 through 11067.

The resulting T-SQL query is immense:

#### **T-SQL**

```
SELECT
[Project2].[OrderID] AS [OrderID],
[Project2].[OrderDate] AS [OrderDate],
[Project2].[CustomerID] AS [CustomerID],
[Project2].[EmployeeID] AS [EmployeeID],
[Project2].[ShipperID] AS [ShipperID],
[Project2].[C1] AS [C1],
[Project2].[C2] AS [C2],
[Project2].[OrderID1] AS [OrderID1],
[Project2].[ProductID] AS [ProductID],
[Project2].[UnitPrice] AS [UnitPrice],
[Project2].[Quantity] AS [Quantity],
[Project2].[Discount] AS [Discount]
FROM ( SELECT
[Limit1].[OrderID] AS [OrderID],
[Limit1].[OrderDate] AS [OrderDate],
[Limit1].[CustomerID] AS [CustomerID],
[Limit1].[EmployeeID] AS [EmployeeID],
[Limit1].[ShipperID] AS [ShipperID],
[Limit1].[C1] AS [C1],
```

```

[Extent5].[OrderID] AS [OrderID1],
[Extent5].[ProductID] AS [ProductID],
[Extent5].[UnitPrice] AS [UnitPrice],
[Extent5].[Quantity] AS [Quantity],
[Extent5].[Discount] AS [Discount],
CASE WHEN ((Extent5).[OrderID] IS NULL)
    THEN CAST(NULL AS int) ELSE 1 END AS [C2]
FROM   (SELECT TOP (5) [Project1].[OrderID] AS [OrderID],
[Project1].[OrderDate] AS [OrderDate], [Project1].[CustomerID] AS [CustomerID],
[Project1].[EmployeeID] AS [EmployeeID], [Project1].[ShipperID] AS [ShipperID],
[Project1].[C1] AS [C1]
FROM  ( SELECT [Project1].[OrderID] AS [OrderID],
[Project1].[OrderDate] AS [OrderDate],
[Project1].[CustomerID] AS [CustomerID],
[Project1].[EmployeeID] AS [EmployeeID],
[Project1].[ShipperID] AS [ShipperID],
[Project1].[C1] AS [C1], row_number()
OVER (ORDER BY [Project1].[OrderID] DESC) AS [row_number]
FROM  ( SELECT
[Extent1].[OrderID] AS [OrderID],
[Extent1].[OrderDate] AS [OrderDate],
[Extent2].[CustomerID] AS [CustomerID],
[Extent3].[EmployeeID] AS [EmployeeID],
[Extent4].[ShipperID] AS [ShipperID],
1 AS [C1]
FROM    [dbo].[Orders] AS [Extent1]
LEFT OUTER JOIN [dbo].[Customers] AS [Extent2]
ON [Extent1].[CustomerID] = [Extent2].[CustomerID]
LEFT OUTER JOIN [dbo].[Employees] AS [Extent3]
ON [Extent1].[EmployeeID] = [Extent3].[EmployeeID]
LEFT OUTER JOIN [dbo].[Shippers] AS [Extent4]
ON [Extent1].[ShipVia] = [Extent4].[ShipperID]
) AS [Project1]
) AS [Project1]
WHERE [Project1].[row_number] > 10
ORDER BY [Project1].[OrderID] DESC ) AS [Limit1]
LEFT OUTER JOIN [dbo].[Order Details] AS [Extent5]
ON [Limit1].[OrderID] = [Extent5].[OrderID]
) AS [Project2]
ORDER BY [Project2].[OrderID] DESC, [Project2].[CustomerID] ASC,
[Project2].[EmployeeID] ASC, [Project2].[ShipperID] ASC, [Project2].[C2] ASC
ORDER BY Country, City;

```

## ***IS OF, OFTYPE, and TREAT Are Type Operators for Polymorphic Queries***

The `expression IS [NOT] OF ([ONLY] type)` operator is similar to C#'s `is` and VB's `Is` operators. If the `ONLY` condition is applied, the operator returns `TRUE` only if `expression` is of the type `type` and not a subtype of `type`; otherwise it returns `TRUE` if `expression` is of the type `type` or one of its subtypes.

The `TREAT expression AS data_type` is similar to C#'s `as` and VB's `TryCast()` operators, which perform conversions between compatible reference types. `TREAT derived_type AS base_type` returns an instance of `base_type`, but `TREAT base_type AS derived_type` returns `NULL`.

## Part IV: Introducing the ADO.NET Entity Framework

OFTYPE(expression, [ONLY] test\_type) returns a collection of the test\_type type from a query expression.

The “Querying Base and Derived Classes” section of Chapter 10 has examples that demonstrate the use of these three operators.

### Subqueries That Return Aggregate Values for WHERE Clause Constraints Throw Exceptions

T-SQL executes the following query to find the last Order entered with no problem:

#### T-SQL

```
SELECT * FROM Orders  
WHERE OrderID = (SELECT MAX(OrderID) FROM Orders);
```

However, executing the corresponding eSQL query:

#### eSQL

```
SELECT VALUE o  
FROM NorthwindEntities.Orders AS o  
WHERE o.OrderID = (SELECT MAX(oMax.OrderID)  
                    FROM NorthwindEntities.Orders AS oMax);
```

throws the exception shown in Figure 11-10.

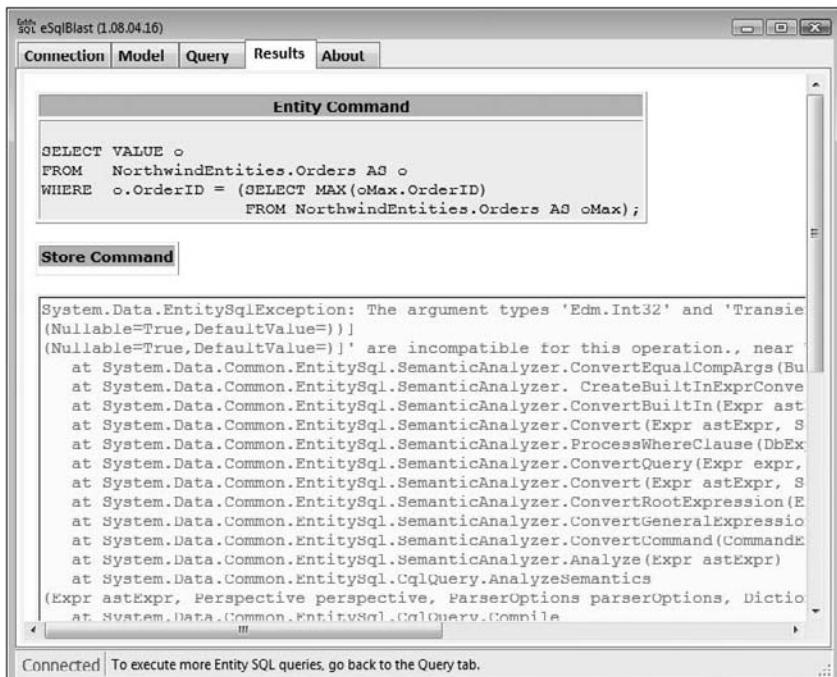


Figure 11-10

*The full text of the exception's message is "System.Data.EntitySqlException: The argument types 'Edm.Int32 and 'Transient.collection[Transient.rowtype{(\_##groupAggMax2,Edm.Int32(Nullable=True, DefaultValue=))}(Nullable=True,DefaultValue=)]' are incompatible for this operation., near WHERE predicate, line 4, column 18."*

The subquery returns a Transient.collection of Transient.rowtype of anonymous type \_##groupAggMax2, Edm.Int32, which eSQL won't convert to an int. T-SQL and other SQL dialects have no compunctions about casting a single column of a single row to an int or other numeric data type.

The obvious workaround is to execute a parameterized query with the value returned by `SELECT MAX(oMax.OrderID) FROM NorthwindEntities.Orders AS oMax` used as the WHERE clause constraint.

## Executing eSQL Queries against the EntityClient

The `System.Data.EntityClient` class is a relatively thin EDM wrapper around the data store's managed ADO.NET client class. The class provides a hierarchy of `EntityConnection`, `EntityCommand` and `EntityDataReader` objects that correspond to the `System.Data.Common` namespace's base `DbConnection`, `DbCommand`, and `DbDataReader` classes. `EntityParameter` and `EntityTransaction` parallel their `System.Data.Common` counterparts also.

The following snippets create and iterate an `Orders` `EntitySet` from the `NorthwindEntities` `EntityContainer`:

### C# 3.0

```
using (EntityConnection econNwind = new EntityConnection("name=NorthwindEntities"))
{
    try
    {
        econNwind.Open();
        using (EntityCommand ecmdNwind = econNwind.CreateCommand())
        {
            string eSql = "SELECT VALUE o FROM NorthwindEntities.Orders AS o " +
                          " ORDER BY o.OrderID DESC LIMIT(5); ";
            ecmdNwind.CommandText = eSql;
            using (EntityDataReader edrNwind =
                  ecmdNwind.ExecuteReader(CommandBehavior.SequentialAccess))
            {
                while (edrNwind.Read())
                {
                    // Do something with the data
                }
            }
        }
    }
}
```

## Part IV: Introducing the ADO.NET Entity Framework

---

```
        catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Northwind eSQL EntityClient Test");
    }
}
```

### VB 9.0

```
Using econNwind As New EntityConnection("name=NorthwindEntities")
    Dim lastESql As String = txtQuery.Text
    Try
        econNwind.Open()
        Using ecmdNwind As EntityCommand = econNwind.CreateCommand()
            Dim eSql As String = "SELECT VALUE o FROM NorthwindEntities.Orders " & _
                " AS o ORDER BY o.OrderID DESC LIMIT(5); "
            ecmdNwind.CommandText = eSql
            Using edrNwind As EntityDataReader = _
                ecmdNwind.ExecuteReader(CommandBehavior.SequentialAccess)
                Do While edrNwind.Read()
                    ' Do something with the data
                Loop
            End Using
        End Using
    Catch ex As Exception
        MessageBox.Show(ex.Message, "Northwind eSQL EntityClient Test")
    End Try
End Using
```

There's no significant difference between the structure of `EntityClient` queries and `SqlClient`, `OracleClient` or other conventional ADO.NET `DbClient` queries.

## Parsing the `IExtendedDataRecord` from an `EntityDataReader`

The conventional approach to parsing an `IDataRecord` returned by a `DbDataReader` is to extract its metadata by iterating the columns with a `Type type = record.GetType(columnIndex)` statement. The `BuiltInTypeKind` enumeration lists 40 built-in types. `EntityType`, `ComplexType`, and `RowType` are EDM *structural types*; structural types derive from the `StructuralType` class and have members. However, the `RefType`, `PrimitiveType`, and `CollectionType` types have a single field per record and don't have a populated `FieldMetadata` collection. Thus, iterating a nested `EntityDataReader` with `fieldCount - 1` as the upper limit would miss these three types.

The eSQLBlast project uses code similar to the preceding C# sample to process `EntityClient` queries. eSQLBlast's `IMultiReaderVisitor` interface and the `XmlVisitor` class that implements it are part of the `Microsoft.Samples.Data.eSqlBlast.Core` project. Methods that implement the *Visitor pattern* enable performing different sets of operations on an object without modifying the object's code. The following recursive `VisitRecord` method of the `XmlVisitor` class creates the XML representation of the data when you mark the Query page's Raw Results check box (refer to Figure 11-6).

**C# 3.0**

```
private void VisitRecord(IExtendedDataRecord record,
                        string elementType, XElement xmlParent)
{
    XElement xmlRecord = _XmlDoc.CreateElement(elementType);
    xmlParent.AppendChild(xmlRecord);

    // First get the type kind of the record itself.
    // A record is not necessarily a structural type.
    BuiltInTypeKind recordTypeKind =
        record.DataRecordInfo.RecordType.EdmType.BuiltInTypeKind;
    int fieldCount;

    // For RefType, PrimitiveType, and CollectionType the
    // FieldMetadata collection is not populated.
    // In those cases the record contains exactly one field.
    if (recordTypeKind == BuiltInTypeKind.RefType ||
        recordTypeKind == BuiltInTypeKind.PrimitiveType ||
        recordTypeKind == BuiltInTypeKind.CollectionType)
    {
        fieldCount = 1;
    }
    else
    {
        fieldCount = record.DataRecordInfo.FieldMetadata.Count;
    }

    for (int i = 0; i < fieldCount; i++)
    {
        string name = record.GetName(i);
        XElement xmlProperty =
            _XmlDoc.CreateElement(XmlConvert.EncodeName(name));
        xmlRecord.AppendChild(xmlProperty);

        // Field flagged as DBNull have undetermined value shapes.
        // An attempt to get such a value may trigger an exception.
        if (record.IsDBNull(i))
        {
            XmlAttribute xmlNull = _XmlDoc.CreateAttribute(_Null);
            xmlProperty.Attributes.Append(xmlNull);
            xmlNull.InnerText = true.ToString();
        }
        else
        {
            BuiltInTypeKind fieldTypeKind = recordTypeKind;
            // If the record is a structural type, examine the field.
            if (recordTypeKind == BuiltInTypeKind.EntityType ||
                recordTypeKind == BuiltInTypeKind.ComplexType ||
                recordTypeKind == BuiltInTypeKind.RowType)
            {
                fieldTypeKind = record.DataRecordInfo.FieldMetadata[i].
```

## Part IV: Introducing the ADO.NET Entity Framework

---

```
        FieldType.TypeUsage.EdmType.BuiltInTypeKind;
    }
    switch (fieldTypeKind)
    {
        case BuiltInTypeKind.CollectionType:
            // Collections are surfaced as nested DbDataReaders.
            VisitReader(record.GetData(i) as DbDataReader,
                        xmlProperty);
            break;

        // EntityType, ComplexType and RowType are wrapped in
        // implementations of IExtendedDataRecord.
        case BuiltInTypeKind.EntityType:
            VisitRecord(record.GetDataRecord(i) as IExtendedDataRecord,
                        _Entity, xmlProperty);
            break;

        case BuiltInTypeKind.ComplexType:
            VisitRecord(record.GetDataRecord(i) as IExtendedDataRecord,
                        _Struct, xmlProperty);
            break;

        case BuiltInTypeKind.RowType:
            VisitRecord(record.GetDataRecord(i) as IExtendedDataRecord,
                        _Row, xmlProperty);
            break;

        case BuiltInTypeKind.RefType:
            // Ref types are surfaced as EntityKey instances.
            // The containing record sees them as atomic.
            VisitKey(record.GetValue(i) as EntityKey, xmlProperty);
            break;

        case BuiltInTypeKind.PrimitiveType:
            // Primitive types are surfaced as plain objects.
            xmlProperty.InnerText = record.GetValue(i).ToString();
            break;

        default:
            // Unhandled type kind. We shouldn't end up here.
            xmlProperty.InnerText = "Unhandled type kind [" +
                fieldTypeKind.ToString() + "]";
            break;
    }
}
}
```

The `XmlVisitor` class's `VisitReader` method and the preceding code can serve as the starting point for delivering `EntityDataReader` output in a variety of XML formats, including RSS or Atom, other common text formats, such as JavaScript Object Notation (JSON).

## Measuring the Performance Penalty of EntitySQL Queries

The NwindEntitySqlCS.sln and NwindEntitySqlVB.sln sample projects in the \WROX\ADONET\Chapter11\CS\ NwindEntitySqlCS and ... \VB\NwindEntitySqlVB folders are simplified versions of the eSQLBlast C# project. These two projects execute arbitrary EntityClient queries against the NorthwindEntities container and display the results with simpler VisitReader() and VisitRecord() methods to generate the XML query result representation. Online Help's "How to: Execute an Entity SQL Query Using EntityCommand" topic includes the VisitReader() and VisitRecord() methods' code. The projects include a Stopwatch timer and an option to repeat the current query 1,000 times to increase timing resolution and accuracy.

A check box enables viewing the T-SQL batch sent to the data store, which enables comparing the performance of the EntityDataReader with the corresponding SqlDataReader when you copy the T-SQL command to the upper query text box. The execution time displayed is for the EntityDataReader or SqlDataReader only; it doesn't include time to open the connection or create the EntityCommand.

Figure 11-11 shows NwindEntitySqlCS.sln displaying the result of executing the "Nested Queries Are Required for One:Many Navigation Properties" section's sample query 1,000 times, 0.942 second. Executing the generated T-SQL 1,000 times requires 0.843 seconds, so translating from eSQL to T-SQL requires about 100 µs. for a moderately fast (2.33 GHz) computer.

```

<?xml version="1.0" encoding="utf-16"?>
<Command>
<Result>
<Record>
<OrderID>10248</OrderID>
<CustID>VINET</CustID>
<EmpID>5</EmpID>
<Date>7/4/1996 12:00:00 AM</Date>
<ShipID>3</ShipID>
<_x0023_x0023_0>
<Result>
<Record>
<Quan.>12</Quan.>
<Category>Dairy Products</Category>
<SKU>11</SKU>
<ProductName>Queso Cabrales</ProductName>

```

**Figure 11-11**

This chapter's sample folders also include AwLiteEntitySqlCS.sln and AwFullEntitySqlCS.sln projects, which use the AdventureWorksLT and AdventureWorks SQL Server 2005 sample databases. The purpose of these four sample projects is to enable you to compare the performance of eSQL and T-SQL queries against larger tables in databases of more realistic size and complexity.

### Executing Parameterized eSQL Queries

Most real-world projects use parameterized eSQL queries because they're resistant to SQL-injection attacks. Parameterized eSQL queries against the Object Layer also let you take advantage of the increased performance of compiled queries for multiple executions with varying parameter values. Code to parameterize eSQL queries is almost identical to that for `DbCommand` queries. In both cases, parameter values require an underlying object type that's compatible with the parameter's `DbType` data type. However, EDM appears to a bit less flexible in matching .NET data types with `DbType` data types.

The following code in the `NwindEntitySqlCS.sln`'s `btnExecute_Click` event handler detects the `@` parameter name prefix in the query typed in or pasted to the query text box and extracts the parameter name. A faux C# text box, based on a `InputDialogDialog` class from the "Create A VB InputBox in C#" article by Les Smith (<http://www.knowdotnet.com/articles/inputbox.html>), accepts user input. A series of `CLRTyp.TryParse()` invocations transform the `InputBox` return string into objects with underlying `int`, `Decimal`, `DateTime`, and `string` data types.

#### C# 3.0

```
if (lastEsql.Contains("@"))
{
    // It's a parameterized query; create a temporary string for parsing
    string tempEsql = lastEsql;
    while (tempEsql.Contains("@"))
    {
        // Get the parameter name from a C# MessageBox implementation
        tempEsql = tempEsql.Substring(tempEsql.IndexOf("@") + 1);
        string prmName = tempEsql.Substring(0, tempEsql.IndexOf(" "));
        string msg = "Please type the value for the '" + prmName +
            "' parameter and click OK.";
        string title = "Entity SQL Test Harness for Northwind";
        string prmValue = InputBox(msg, title, null);
        if (prmValue != null)
        {
            // Detect type to prevent "Argument types 'Edm.Int32' and 'Edm.String'"
            // are incompatible for this operation, ..." exceptions
            int testInt = 0;
            decimal testDec = 0M;
            DateTime testDate = DateTime.MinValue;
            if (int.TryParse(prmValue, out testInt))
                ecmdNwind.Parameters.AddWithValue(prmName, (object)testInt);
            else if (Decimal.TryParse(prmValue, out testDec))
                ecmdNwind.Parameters.AddWithValue(prmName, (object)testDec);
            else if (DateTime.TryParse(prmValue, out testDate))
                ecmdNwind.Parameters.AddWithValue(prmName, (object)testDate);
            else
                ecmdNwind.Parameters.AddWithValue(prmName, (object)prmValue);
            tempEsql = tempEsql.Substring(tempEsql.IndexOf(" ") + 1);
        }
    }
}
```

The VB version of the preceding C# method uses VB's native InputBox control.

### VB 9.0

```
If lastEsql.Contains("@") Then
    ' It's a parameterized query
    Dim tempEsql As String = lastEsql
    Do While tempEsql.Contains("@")
        tempEsql = tempEsql.Substring(tempEsql.IndexOf("@") + 1)
        Dim prmName As String = tempEsql.Substring(0, tempEsql.IndexOf(" "))
        Dim msg As String = "Please type the value for the '" & prmName & _
            "' parameter and click OK."
        Dim title As String = "Entity SQL Test Harness for Northwind"
        ' Get the parameter value from a VB InputBox
        Dim prmValue As String = InputBox(msg, title, Nothing)
        If prmValue IsNot Nothing Then
            ' Detect type to prevent "Argument types 'Edm.Int32' and 'Edm.String'
            ' are incompatible for this operation, ... " exceptions
            Dim testInt As Integer = 0
            Dim testDec As Decimal = 0D
            Dim testDate As DateTime = DateTime.MinValue
            If Integer.TryParse(prmValue, testInt) Then
                ecmdNwind.Parameters.AddWithValue(prmName, CObj(testInt))
            ElseIf Decimal.TryParse(prmValue, testDec) Then
                ecmdNwind.Parameters.AddWithValue(prmName, CObj(testDec))
            ElseIf DateTime.TryParse(prmValue, testDate) Then
                ecmdNwind.Parameters.AddWithValue(prmName, CObj(testDate))
            Else
                ecmdNwind.Parameters.AddWithValue(prmName, CObj(prmValue))
            End If
            tempEsql = tempEsql.Substring(tempEsql.IndexOf(" ") + 1)
        End If
    Loop
End If
```

Both sample project versions include a `paramQuery` string to supply a parameterized query for testing `int` and `string` data types. If parameters are named with property names, it's not a major task to iterate the columns with an `EntityDataReader.GetType(i)` method, cache the data types in a `Dictionary`, and expand the type conversion code's repertoire.

## Using SQL Server Compact as an Entity Framework Data Store

SQL Server Compact (Edition) v3.5 (SSCE, also called Compact) is an alternative data source for EF and LINQ to SQL. SSCE is a single-user, embedded database engine with a very small memory footprint (about 2 MB) that Microsoft's licensing terms originally restricted to mobile devices and tablet PCs. Microsoft now licenses SSCE v3.5 for use on desktop and tablet PCs or mobile devices at no charge. SSCE often serves as a high-performance, client-side data cache and is a common replication target for ADO.NET Synchronization Services, Microsoft Synchronization Framework, and Live Mesh. Password protection and encryption of data files is optional. SSCE supports a subset of T-SQL, and you can open

## Part IV: Introducing the ADO.NET Entity Framework

existing or create new \*.sdf data files with VS 2008's Server Explorer, VS 2008 Express's Database Explorer or SQL Server Management Studio [Express] 2008.

SSCE is a file-based database engine that doesn't permit simultaneous access by multiple users. However, an application can open multiple connections to a \*.sdf file. The \*.sdf file must be on the same machine as the application that connects to it. SSCE doesn't support networked data files, or stored views or procedures.

### **Substituting SSCE for SQL Server [Express] as a Data Store**

VS 2008 SP1 installs a version of SSCE and provides a Northwind.sdf file that's compatible with EF in the \Program Files\Microsoft SQL Server Compact Edition\v3.5\Samples folder. Substituting SSCE for SQL Server 2005+ [Express] requires specifying *FileName*.sdf instead of an SQL Server instance when you select the Generate from Database option in the Wizard's Choose Model Contents dialog and click next. Following are the changes to the data source selection process:

1. Click the New Connection button to open the Connection Properties dialog.
2. Click the Change in the Data Source button to open the Change Data Source dialog.
3. Select Microsoft SQL Server Compact 3.5, and click OK to close the dialog.
4. With My Computer selected as the Data Source, click the Browse button and navigate to the \*.sdf data source, and click OK to close the file dialog (see Figure 11-12).



Figure 11-12

5. The Wizard doesn't prefix the connection string or model namespace name with the filename, so change the connection string name to `DomainNameEntities`, and click Next.
6. Accept the Tables selection, change the Model Namespace to `DomainNameModel`, and click Finish to generate the `*.edmx` file and the `DomainNameModel` namespace's classes.
7. Double-click the `App.config` or `Web.config` file in Solution Explorer to open it and change the provider entry from `provider=Microsoft.SqlServerCe.Client.3.5` to `provider=System.Data.SqlServerCe.3.5`.
8. If you want to create a ADO.NET connection to SSCE, add a reference to `System.Data.SqlServerCe` and using `System.Data.SqlServerCe;` or Imports `System.Data.SqlServerCe` directive to your class file.
9. If you store the database file in the `...\\AppFolder\\bin\\Debug` folder and use the Connection Properties dialog to create the connection string, remove the extraneous `\\bin\\debug` path from `|DataDirectory|\\bin\\Debug`. The correct path is `|DataDirectory|DatabaseName.sdf`.

*For reasons known only to Microsoft India's SSCE v3.5 developers, the column names of the sample Northwind.sdf file in the \Program Files\Microsoft SQL Server Compact Edition\v3.5\Samples folder include spaces. (The last Microsoft sample database with spaces in field names was Access 2.0, which debuted in 1994.) Therefore, this chapter's SSCE sample projects use a custom version of Northwind.sdf created from a modified version of the InstNwnd.sql script included with the SQL Server 2000 Sample Databases download. The custom version in the \WROX\ADONET\Chapter11\NorthwindSSCE folder is identical to the SQL Server 2005+ [Express] instance from the Northwind.mdf file used for this book's other sample programs.*

## Summary

eSQL is a SQL dialect that includes extensions to accommodate querying .NET domain objects (entities) rather than relational tables. Extensions include support for many:one, one:many, and many:many associations, as well as common inheritance models. However, eSQL doesn't support DDL for creating, modifying, or dropping database objects or DML for executing INSERT, UPDATE, and DELETE operations.

This chapter is primarily devoted to describing differences between eSQL and other ANSI SQL dialects, specifically the T-SQL query language of SQL Server 2005 and later. Demonstrating these SQL extensions requires executing queries containing them and viewing the resultsets, so the chapter began with a description of Zlatko Michalev's eSQLBlast eSQL query analyzer tool. eSQLBlast displays entities retrieved from the data store as hierarchical XML documents or nested HTML tables. As is the case for this chapter's other sample applications, the data store is a customized version of the Northwind sample SQL Server 2000 database. eSQL and T-SQL examples demonstrate syntax that diverges significantly.

eSQLBlast and this chapter's sample applications use `EntityConnection`, `EntityCommand` and `EntityDataReader` objects to execute queries against the `EntityClient` layer. These `EntityClient` objects, which emulate ADO.NET's traditional `SqlClient` objects, have a relatively thin `EntityClient` wrapper to translate eSQL to conventional T-SQL queries. Test harnesses that execute simple queries against Northwind, AdventureWorksLT, and AdventureWorks data sources demonstrate that simple query translations require about 100 µs. The Northwind test harnesses have an added feature to demonstrate parameterized queries.

## Part IV: Introducing the ADO.NET Entity Framework

---

SQL Server Compact Edition (SSCE) v3.5 is undergoing widening deployment as an embedded database engine to enable caching and persisting relatively large amounts of data for occasionally connected clients. Both LINQ to SQL and EF v1's Entity Data Model (EDM) support SSCE as a data source, but EF treats SSCE as a first-class citizen, similarly to SQL Server [Express] 2005+, while LINQ to SQL v1's graphical O/R Designer doesn't support SSCE. C# and VB test harnesses for a customized SSCE Northwind.sdf database fill out this chapter's repertoire of sample projects. Although EDM overwhelms SSCE's 2-MB memory footprint, SSCE is expected to gain acceptance in smart-client applications, such as Live Mesh projects that consume and cache information from the new databases in the cloud. EF presently is the preferred persistence provider for ADO.NET Data Services (Astoria) and is likely to gain the same status with SQL Server Data Services (SSDS) as the SSDS URL query syntax and data transfer protocol become aligned with Astoria.

# **Part V: Implementing the ADO.NET Entity Framework**

**Chapter 12:** Taking Advantage of Object Services and LINQ  
to Entities

**Chapter 13:** Updating Entities and Complex Types

**Chapter 14:** Binding Entities to Data-Aware Controls

**Chapter 15:** Using the Entity Framework as a Data Source

## Part V: Implementing the ADO.NET Entity Framework

---

Entity Framework (EF) v1 is the first and — when this book was written — the only concrete implementation of the Entity Data Model (EDM). EF's owner is Microsoft's SQL Server Data Programmability (DP) group, which is responsible for the entire ADO.NET technology stack, including ADO.NET Data Services, ADO.NET Synchronization Services, and SQL Server Data Services. EF competes directly with LINQ to SQL, which was developed primarily by members of the ObjectSpaces team, initially Matt Warren and Luca Bolognese of the C# group.

*Matt Warren's The Origin of LINQ to SQL blog post of May 31, 2007 (<http://blogs.msdn.com/mattwar/archive/2007/05/31/the-origin-of-linq-to-sql.aspx>) recounts the demise of ObjectSpaces in the black hole of WinFS and describes his three years after the event as a "political nightmare."*

DP gained control of the LINQ to SQL implementation in October 2007, shortly before VS 2008 released to manufacturing. The DP group positions LINQ to SQL, which shipped with the original VS 2008 release, as a rapid application development (RAD) O/RM tool for small to medium-sized business applications. DP product managers claim EF is an enterprise-grade product that's capable of mapping complex object constructs, such as multiple inheritance types, value types (which EF calls *complex types*), and many:many associations (relationships), that LINQ to SQL doesn't support.

## Entity Framework vs. LINQ to SQL

Offering two competing O/RM tools with different feature sets to .NET developers is difficult to justify as an ongoing business strategy, and the DP's team's large-scale investment in and grandiose plans for EF made orphaning of LINQ to SQL inevitable.

Tim Mallalieu, program manager for LINQ to SQL and Entity Framework, posted the "Update on LINQ to SQL and LINQ to Entities Roadmap" article to the ADO.NET Team Blog on October 29, 2008 (<http://blogs.msdn.com/adonet/archive/2008/10/29/update-on-linq-to-sql-and-linq-to-entities-roadmap.aspx>). Following is the article's last paragraph:

*We're making significant investments in the Entity Framework such that as of .NET 4.0 the Entity Framework will be our recommended data access solution for LINQ to relational scenarios. We are listening to customers regarding LINQ to SQL and will continue to evolve the product based on feedback we receive from the community as well.*

Most authors of data-intensive VS 2008 project examples — such as Scott Guthrie, corporate VP of the .NET Developer Division — used LINQ to SQL as the preferred data source. Guthrie wrote a detailed, nine-part LINQ to SQL tutorial targeted to ASP.NET and Web developers in 2007 (<http://weblogs.asp.net/scottgu/archive/2007/09/07/linq-to-sql-part-9-using-a-custom-linq-expression-with-the-lt-asp-linqdatasource-gt-control.aspx>). Sample ASP.NET Model-View-Controller (MVC) and ASP.NET Dynamic Design projects used LINQ to SQL rather than EF as their data source. This isn't surprising when you consider that LINQ to SQL was a fully qualified component of .NET 3.5 and EF didn't arrive until .NET 3.5 and VS 2008 SP1.

Many developers had made significant investments in applying LINQ to SQL to new .NET 3.5 projects before EF was released. Scott Hanselman, Microsoft's principal program manager for community liaison, released in October 2008 an "Informal .NET Subsystem Survey" that displayed the number of developers using each of 14 .NET 3.5 and VS 2008 SP1 features (<http://www.hanselman.com/blog/SurveyRESULTSWhatNETFrameworkFeaturesDoYouUse.aspx>). Of 4,899 respondents in a one-week

period, 1,734 (35.5%) reported using LINQ to SQL versus 643 (13.1%) using EF. Use of LINQ to SQL was almost neck-and-neck with ADO.NET DataSets at 1,887 users (38.5%).

Therefore, it's not surprising that Mallalieu's initial blog post engendered an outcry from the .NET developer community. An extraordinary number of apocalyptic "Death of LINQ to SQL" or similarly titled posts appeared on developer-oriented blogs. Rumors of LINQ to SQL's death are premature, because its assemblies are part of the .NET 3.5 [SP1] System.Data namespace and must be retained *ad infinitum* for backward compatibility.

Mallalieu attempted to defend the DP group's position in an October 31, 2008 "Clarifying the message on L2S Features" post (<http://blogs.msdn.com/adonet/archive/2008/10/31/clarifying-the-message-on-l2s-futures.aspx>) attempted to deflect the uproar by asking the rhetorical question, "Is LINQ dead?" He then asserted that "we decided to take the EF forward with regards to the overall convergence effort and over time provide a single solution that can address the various asks" and confirmed EF's status as the "recommended data access solution for LINQ to relational scenarios."

The upshot of the DP group's decision is that LINQ to SQL will enter the purgatory of deprecated technologies upon the release of .NET 4.0 and VS 2010. Microsoft-supported improvements to LINQ to SQL will be few and far between until then and non-existent thereafter. This means developers should place bets on EF for data-intensive projects other than those for short-term, ad hoc individual or departmental application.

## Entity Framework Futures

Tim Mallalieu's "Entity Framework Futures" session for the Microsoft Professional Developers Conference (PDC) 2008 outlined in more detail what developers can expect from EF v2 and other future Entity Data Model implementations. (You can view the Channel9 video of the presentation at <http://channel9.msdn.com/pdc2008/TL20/>).

Mallalieu emphasized that data access is the first EDM implementation, but the Entity Data Platform (EDP) will ultimately manage the following domains:

- Data access and object persistence (EF)
- Integration, aggregation, and synchronization
- Reporting and analytics
- Data management, deployment, policy, and security
- Models and workflows

These scenarios will be supported by four primary EDP technology categories:

- Client technologies
- Mid-tier frameworks and technologies
- Data access APIs and frameworks
- Core storage

## Part V: Implementing the ADO.NET Entity Framework

---

Data consumers fall into the following segments, each with their own needs in the preceding domains and technologies:

- Traditional data developers
- Developers using scripting and “rebel frameworks,” such as agile programming
- Model-centric data developers
- Line of business (LOB) script or macro writers
- Information workers

To satisfy agile programmers who insist on plain old CLR objects (POCOs) and total persistence ignorance, Mallalieu demonstrated writing class code for POCOs and an `ObjectContext` in the simplest possible, convention-oriented way. This approach eliminates the EDM and its XML files, and defines an `ObjectSet` that implements `IQueryable` to deliver completely persistence ignorant code. Adding mapping attributes enables configuration-oriented code to specify mapping to tables in other schemas or with column names that don't correspond to property names.

Model-centric developers want to define a model with the EDM Designer and then generate the database from the model in a manner similar to that employed by LINQ to SQL. EF v2 will add this capability, which (unfortunately) destroys all data in the database, as well as the capability to define complex types and implement them by a new “Artifact Generation Pipeline” that combines Windows Workflow and T4 templates. This feature enables developers to modify the workflow, customize the templates, and build their own design tooling.

Framework developers are independent software vendors (ISVs) who want to build out a data access layer that leverages the EDM. For example a DBA might create the EDM and the framework developer would use the `*.edmx` model to generate POCO classes and enable lazy loading, which was missing from v1. To prevent the need for direct access to database tables, EF v2 will enable SQL Server table-valued functions as a data source for composable LINQ to SQL queries, which stored procedures don't support. New model-defined functions (MDFs) let developers add function definitions, such as `FullName()`, to the model. You define MDFs in code, add the SQL expression in the `*.edmx` file; the expression tree executes the expression in-line.

Finally, a new feature nicknamed “Slambda” lets you use a `DbExpression` or comprehension syntax to build a query tree which returns a ADO.NET `DbDataReader` instead of a typed collection. Beyond v2, EF and EDM will become a part of the forthcoming Oslo modeling framework with an `MEntity` domain-specific grammar. Missing from the presentation was an example of *n*-tier deployment of EF without the added overhead of an ADO.NET Data Services (Astoria) layer, as well as support for `Contains` and enums, which the DP team won't commit to including in v2.

# 12

## Taking Advantage of Object Services and LINQ to Entities

Executing Entity SQL (eSQL) queries against the read-only `EntityClient` layer is a good way of learning the differences between eSQL, and T-SQL query syntax, but the vast majority of real-world Entity Framework (EF) projects will take advantage of the Object Services API. Chapter 9 provided brief examples for using the Object Services API's primary classes: `ObjectContext` and its `ObjectQuery<T>`, `ObjectParameter`, `ObjectResult<T>`, `ObjectStateManager`, and `ObjectStateEntry` types. The Object Services API enables create, retrieve, update, and delete (CRUD) operation on the tables of the underlying data store with strongly typed CLR objects that you define by the mapping files of the Entity Data Model (EDM). The `System.Data.Objects` and `System.Data.Objects.DataClasses` namespaces — both of which are included in the `System.Data.Entity` assembly — contain the Object Services API's classes.

`ObjectContext` is EF's top-level object. `ObjectContext`'s primary job is identifying objects and managing cached object graphs in local memory, tracking changes to objects, and persisting the changes to the data store. In other words, `ObjectContext` is the "tool" component of EF as an object/relational mapping (O/RM) tool. `ObjectContext` encapsulates `MetadataWorkspace` and `EntityConnection` objects, which it shares with `EntityClient`; Chapters 10 and 11 also introduced you to these two objects. The `ObjectContext`'s `ObjectStateManager` is responsible for identifying entity instances and keeping track of changes to entity property values.

*ObjectContext performs services similar to those of LINQ to SQL's `DataContext` but otherwise has little in common with the latter object. For example, `ObjectContext` doesn't support lazy loading, which EF calls deferred loading. Neither `ObjectContext` nor `DataContext` is serializable for passing between tiers of a distributed EF deployment. However, EF can pass `EntitySets` with both many:one and one:many associations between tiers, but LINQ to SQL v1 can't without the aid of a third-party add-in. The "Entity Class Serialization" section describes `EntitySet` serialization in more detail.*

# Exploring the Generated Entity Classes

The `ModelName.Designer.cs` or `.vb` file defines the `ModelNameModel` namespace, which is decorated by a single `EdmSchemaAttribute` and an `EdmRelationshipAttribute` for each association. These attributes are applied at the assembly level, as indicated by the `assembly:` prefix; they indicate that the assembly contains entity data classes and define the role for each entity in the particular association. Following is the syntax for the `EdmRelationshipAttribute` class:

### C# 3.0

```
public EdmRelationshipAttribute(
    string relationshipNamespaceName,
    string relationshipName,
    string role1Name, RelationshipMultiplicity role1Multiplicity, Type role1Type,
    string role2Name, RelationshipMultiplicity role2Multiplicity, Type role2Type)
```

### VB 9.0

```
Public Sub New (relationshipNamespaceName As String, _
    relationshipName As String, _
    role1Name As String, role1Multiplicity As RelationshipMultiplicity, _
    role1Type As Type, _
    role2Name As String, role2Multiplicity As RelationshipMultiplicity, _
    role2Type As Type)
```

The `ModelName.csdl` file provides the argument values.

Here's an example of an `EdmRelationshipAttribute` for a one:many association between the Northwind `Customers` and `Orders` EntitySets:

### C# 3.0

```
[assembly: global::System.Data.Objects.DataClasses.EdmRelationshipAttribute
("NorthwindModel", "FK_Orders_Customers", "Customers",
global::System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
typeof(NorthwindModel.Customer), "Orders",
global::System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
typeof(NorthwindModel.Order))]
```

### VB 9.0

```
< ... Assembly:
Global.System.Data.Objects.DataClasses.EdmRelationshipAttribute("NorthwindModel",
    "FK_Orders_Customers", "Customers",
    Global.System.Data.Metadata.Edm.RelationshipMultiplicity.ZeroOrOne,
    GetType(NorthwindModel.Customer), "Orders",
    Global.System.Data.Metadata.Edm.RelationshipMultiplicity.Many,
    GetType(NorthwindModel.Order)), ...
... >
```

### **ModelNameEntities Partial Classes**

The namespace contains code to define a *ModelNameEntities* partial class that's generated from the *ModelName.csdl* file, derives from *ObjectContext* and has a read-only *EntityTypeSetName* property and an *AddToEntityType()* method for each *EntityType* in the model.

Following is an abbreviated and slightly edited version of the *NorthwindEntities* class derived from *ObjectContext*, which returns *ObjectQuery<EntityType>* instances of collections:

#### **C# 3.0**

```
public partial class NorthwindEntities : global::System.Data.Objects.ObjectContext
{
    // Constructor for same-named connection string in App.config
    public NorthwindEntities() :
        base("name=NorthwindEntities", "NorthwindEntities")
    {
    }
    // Constructor for a supplied connection string.
    public NorthwindEntities(string connectionString) :
        base(connectionString, "NorthwindEntities")
    {
    }
    // Constructor for a supplied EntityConnection object.
    public NorthwindEntities(global::
        System.Data.EntityClient.EntityConnection connection) :
        base(connection, "NorthwindEntities")
    {
    }
    [global::System.ComponentModel.BrowsableAttribute(false)]
    public global::System.Data.Objects.ObjectQuery<Category> Categories
    {
        get
        {
            if ((this._Categories == null))
            {
                this._Categories = base.CreateQuery<Category>("[Categories]");
            }
            return this._Categories;
        }
    }
    // Customers, Employees, Order_Details, Orders, Products, Shippers,
    Suppliers
    private global::System.Data.Objects.ObjectQuery<Category> _Categories;
    public void AddToCategories(Category category)
    {
        base.AddObject("Categories", category);
    }
    // Customers, Employees, Order_Details, Orders, Products, Shippers,
    Suppliers
}
```

## Part V: Implementing the ADO.NET Entity Framework

---

### VB 9.0

```
Partial Public Class NorthwindEntities
    Inherits Global.System.Data.Objects.ObjectContext
    ' Constructor for same-named connection string in App.config
    Public Sub New()
        MyBase.New("name=NorthwindEntities", "NorthwindEntities")
    End Sub
    ' Constructor for a supplied connection string.
    Public Sub New(ByVal connectionString As String)
        MyBase.New(connectionString, "NorthwindEntities")
    End Sub
    ' Constructor for a supplied EntityConnection object.
    Public Sub New(ByVal connection As _
        Global.System.Data.EntityClient.EntityConnection)
        MyBase.New(connection, "NorthwindEntities")
    End Sub

    <Global.System.ComponentModel.BrowsableAttribute(false)> _
    Public ReadOnly Property Categories() _
        As Global.System.Data.Objects.ObjectQuery(Of Category)
        Get
            If (Me._Categories Is Nothing) Then
                Me._Categories = MyBase.CreateQuery(Of Category)("[Categories]")
            End If
            Return Me._Categories
        End Get
    End Property
    ' Customers, Employees, Order_Details, Orders, Products, Shippers,
    Suppliers

    Private _Categories As Global.System.Data.Objects.ObjectQuery(Of Category)
    Public Sub AddToCategories(ByVal category As Category)
        MyBase.AddObject("Categories", category)
    End Sub
    ' Customers, Employees, Order_Details, Orders, Products, Shippers,
    Suppliers

End Class
```

ObjectQuery<*EntityType*> collections serve as the data sources for composable eSQL, QueryBuilder, and LINQ to Entities queries. ObjectQuery<*EntityType*> queries are composable because the output type of an ObjectQuery<*EntityType*> is another ObjectQuery<*EntityType*>.

## EntityName Partial Classes

The namespace also contains code for an *EntityName* partial class of the ObjectQuery(<*EntityName*>) type that derives from EntityObject. Decorating *EntityName* classes are an EdmEntityTypeAttribute to provide NamespaceName and table Name data, as well as a DataContractAttribute to mark the class as serializable by the WCF DataContractSerializer. WCF DataContract serialization is opt-in, so a DataMemberAttribute decorates the properties of each serializable *EntityName* class.

## Chapter 12: Taking Advantage of Object Services and LINQ to Entities

Following are abbreviated examples of the Order class with `DataContractAttribute` and `DataMemberAttribute` highlighted:

### C# 3.0

```
[global::System.Data.Objects.DataClasses.EdmEntityTypeAttribute(NamespaceName=
"NorthwindModel", Name="Order")]
[global::System.Runtime.Serialization.DataContractAttribute(IsReference=true)]
[global::System.Serializable()]
public partial class Order :
    global::System.Data.Objects.EntityObject
{
    public static Order CreateOrder(int orderID)
    {
        Order order = new Order();
        order.OrderID = orderID;
        return order;
    }
    [global::System.Data.Objects.DataClasses.
        EdmScalarPropertyAttribute(EntityKeyProperty=true, IsNullable=false)]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    public int OrderID
    {
        get
        {
            return this._OrderID;
        }
        set
        {
            this.OnOrderIDChanging(value);
            this.ReportPropertyChanging("OrderID");
            this._OrderID = global::System.Data.Objects.
                DataClasses.StructuralObject.SetValidValue(value);
            this.ReportPropertyChanged("OrderID");
            this.OnOrderIDChanged();
        }
    }
    private int _OrderID;
    partial void OnOrderIDChanging(int value);
    partial void OnOrderIDChanged();
    // OrderDate ... ShipCountry with DataMemberAttribute

    [global::System.Data.Objects.DataClasses.EdmRelationshipNavigationPropertyAttribute
        ("NorthwindModel", "FK_Orders_Customers", "Customers")]
    [global::System.Xml.Serialization.XmlIgnoreAttribute()]
    [global::System.Xml.Serialization.SoapIgnoreAttribute()]
    [global::System.Runtime.Serialization.DataMemberAttribute()]
    [global::System.ComponentModel.BrowsableAttribute(false)]
    public Customer Customer
```

## Part V: Implementing the ADO.NET Entity Framework

```
{  
    get  
    {  
        return ((global::System.Data.Objects.DataClasses.  
            IEntityWithRelationships)(this)).RelationshipManager.  
            GetRelatedReference<Customer>("NorthwindModel.FK_Orders_Customers",  
            "Customers").Value;  
    }  
    set  
    {  
        ((global::System.Data.Objects.DataClasses.IEntityWithRelationships)  
            (this)).RelationshipManager.GetRelatedReference<Customer>  
            ("NorthwindModel.FK_Orders_Customers", "Customers").Value = value;  
    }  
}  
}  
[global::System.Runtime.Serialization.DataMemberAttribute()]  
[global::System.ComponentModel.BrowsableAttribute(false)]  
public global::System.Data.Objects.DataClasses.  
    EntityReference<Customer> CustomerReference  
{  
    get  
    {  
        return ((global::System.Data.Objects.DataClasses.  
            IEntityWithRelationships)(this)).RelationshipManager.  
            GetRelatedReference<Customer>("NorthwindModel.FK_Orders_Customers",  
            "Customers");  
    }  
    set  
    {  
        if ((value != null))  
        {  
            ((global::System.Data.Objects.DataClasses.  
                IEntityWithRelationships)(this)).RelationshipManager.  
                InitializeRelatedReference<Customer>  
                ("NorthwindModel.FK_Orders_Customers", "Customers", value);  
        }  
    }  
}  
// EmployeeReference and ShipperReference with DataMemberAttribute  
  
[global::System.Data.Objects.DataClasses.  
    EdmRelationshipNavigationPropertyAttribute("NorthwindModel",  
    "FK_Order_Details_Orders", "Order_Details")]  
[global::System.Xml.Serialization.XmlIgnoreAttribute()]  
[global::System.Xml.Serialization.SoapIgnoreAttribute()]  
[global::System.Runtime.Serialization.DataMemberAttribute()]  
[global::System.ComponentModel.BrowsableAttribute(false)]  
public global::System.Data.Objects.DataClasses.
```

## Chapter 12: Taking Advantage of Object Services and LINQ to Entities

```
        EntityCollection<Order_Detail> Order_Details
    {
        get
        {
            return ((global::System.Data.Objects.DataClasses.
                IEntityWithRelationships)(this)).RelationshipManager.
                GetRelatedCollection<Order_Detail>
                ("NorthwindModel.FK_Order_Details_Orders", "Order_Details");
        }
    }
}
```

The Order class is used as an example because it has both many:one (Customers:Orders, Employees:Orders and Shippers:Orders) and one:many (Orders:Order\_Details) associations. The DataContractAttribute's IsReference=true attribute indicates to theDataContractSerializer that associations with Order instances might exist and the DataContractSerializer.PreserveReferences property value should be set to true.

### VB 9.0

```
<Global.System.Data.Objects.DataClasses.EdmEntityTypeAttribute _  
    (NamespaceName:="NorthwindModel", Name:="Order"), _  
    Global.System.Runtime.Serialization.DataContractAttribute(), _  
    Global.System.Serializable()> _  
Partial Public Class Order  
    Inherits Global.System.Data.Objects.DataClasses.EntityObject  
    Public Shared Function CreateOrder(ByVal orderID As Integer) As Order  
        Dim order As Order = New Order  
        order.OrderID = orderID  
        Return order  
    End Function  
  
<Global.System.Data.Objects.DataClasses.EdmScalarPropertyAttribute _  
    (EntityKeyProperty:=true, IsNullable:=false), _  
    Global.System.Runtime.SerializationDataMemberAttribute()> _  
Public Property OrderID() As Integer  
    Get  
        Return Me._OrderID  
    End Get  
    Set  
        Me.OnOrderIDChanging(value)  
        Me.ReportPropertyChanging("OrderID")  
        Me._OrderID = Global.System.Data.Objects.DataClasses _  
            .StructuralObject.SetValidValue(value)  
        Me.ReportPropertyChanged("OrderID")  
        Me.OnOrderIDChanged  
    End Set  
End Property  
  
Private _OrderID As Integer  
  
Partial Private Sub OnOrderIDChanging(ByVal value As Integer)
```

## Part V: Implementing the ADO.NET Entity Framework

---

```
End Sub

Partial Private Sub OnOrderIDChanged()
End Sub

' OrderDate ... ShipCountry with DataMemberAttribute omitted for brevity

<Global.System.Data.Objects.DataClasses. _
  EdmRelationshipNavigationPropertyAttribute("NorthwindModel", _
  "FK_Orders_Customers", "Customers"), _ 
  Global.System.Xml.Serialization.XmlIgnoreAttribute(), _ 
  Global.System.Xml.Serialization.SoapIgnoreAttribute(), _ 
  Global.System.Runtime.Serialization.DataMemberAttribute()> _ 
  Global.System.ComponentModel.BrowsableAttribute(false)> _ 

Public Property Customers() As Customer
Get
  Return CType(Me, Global.System.Data.Objects.DataClasses. _
    IEntityWithRelationships).RelationshipManager. _
    GetRelatedReference(Of Customer) _ 
    ("NorthwindModel.FK_Orders_Customers", "Customers").Value
End Get
Set
  CType(Me, Global.System.Data.Objects.DataClasses. _
    IEntityWithRelationships).RelationshipManager. _
    GetRelatedReference(Of Customer) _ 
    ("NorthwindModel.FK_Orders_Customers", "Customers").Value = value
End Set
End Property

<Global.System.Runtime.Serialization.DataMemberAttribute(), _ 
  Global.System.ComponentModel.BrowsableAttribute(false)> _ 

Public Property CustomersReference() As _
  Global.System.Data.Objects.DataClasses.EntityReference(Of Customer)
Get
  Return CType(Me, Global.System.Data.Objects.DataClasses. _
    IEntityWithRelationships).RelationshipManager.GetRelatedReference _ 
    (Of Customer) ("NorthwindModel.FK_Orders_Customers", "Customers")
End Get
Set
  If (Not (value) Is Nothing) Then
    CType(Me, Global.System.Data.Objects.DataClasses. _
      IEntityWithRelationships).RelationshipManager. _
      InitializeRelatedReference _ 
      (Of Customer) ("NorthwindModel.FK_Orders_Customers", _ 
        "Customers", value)
  End If
End Set
End Property

' EmployeesReference and ShippersReference with DataMemberAttribute omitted

<Global.System.Data.Objects.DataClasses. _
```

## Chapter 12: Taking Advantage of Object Services and LINQ to Entities

```
EdmRelationshipNavigationPropertyAttribute _  
("NorthwindModel", "FK_Order_Details_Orders", "Order_Details"), _  
Global.System.Xml.Serialization.XmlIgnoreAttribute(), _  
Global.System.Xml.Serialization.SoapIgnoreAttribute(), _  
Global.System.Runtime.Serialization.DataMemberAttribute()> _  
Global.System.ComponentModel.BrowsableAttribute(false)> _  
Public ReadOnly Property Order_Details() _  
As Global.System.Data.Objects.DataClasses.EntityCollection(Of Order_Detail)  
Get  
    Return CType(Me, Global.System.Data.Objects.DataClasses. _  
IEntityWithRelationships).RelationshipManager.GetRelatedCollection _  
(Of Order_Detail)("NorthwindModel.FK_Order_Details_Orders", _  
"Order_Details")  
End Get  
End Property  
End Class
```

The preceding classes define the `EntityType` data types for eSQL, QueryBuilder, and LINQ to Entities queries.

### Entity Class Serialization

Object serialization to a data transport format that can pass between processes of distributed systems is required to use EF in *n*-tier, service-oriented architecture (SOA). Object serialization also is required to save object state in `ViewState` between postbacks of ASP.NET Web pages. .NET 3.5 supports the following types of serializers:

1. .NET `BinarySerializer`, which is used to store objects in the `ASP.NET Page.ViewState` property, can save the entity and its related objects on postback, then retrieve the objects during the succeeding page load and attach them to a newly created `ObjectContext`. The `BinarySerializer` also is used for .NET Remoting.
2. .NET `XmlSerializer` for SOAP Web services prior to the release of WCF with .NET 3.0. The `xmlSerializer` can't serialize relationships, which makes it useless for serializing `EntityType`s with relationships.
3. WCF `DataContractSerializer` (DCS) for .NET 3.5 and VS 2008 services. The original DCS version released with .NET 3.5 and VS 2008 could not serialize objects with both many:one and one:many associations because the combination creates a *cyclic relationship* or *cycle*. A relatively minor change in .NET 3.5 Service Pack (SP) 1 enables DCS to handle potential cycles and serialize entire object graphs by a `DataContractAttribute.IsReference=true` attribute value. This feature eliminates the need to write complex hacks to pass `EntityType`s with both types of associations between tiers.
4. WCF `DataContractJsonSerializer` (DCJS) for .NET 3.5 and VS 2008 AJAX services, which has limitations similar to the `XmlSerializer`. DCJS isn't intended for use with `EntityType`s. Using the DCJS with `EntityType`s throws an error due to addition of namespace prefixes to references added by the `DataContractAttribute.IsReference=true` attribute value.

*LINQ to SQL v1 offers serialization of only one:many (unidirectional) associations, so it doesn't preserve EDM object graphs in which any associations are bidirectional. A third-party workaround*

## Part V: Implementing the ADO.NET Entity Framework

---

*described in Chapter 5's "Moving the LINQ to SQL Layer to a Middle Tier" section is available to enable bidirectional serialization.*

### **Serialization with Deferred-Loaded Associated Entities**

If you defer-load associated entities with `IsLoaded()` tests that selectively invoke `Load()` methods in the `foreach` loop, as described in the later "Deferred Loading with the Load() Method" section, the `DataContractSerializer` serializes `CustomerReference`, `EmployeeReference`, and `ShipperReference` properties as `[xs]i:nil="true"` references, rather than objects and ignores references to the associated `Order_Details` `EntitySet`. Therefore, you should avoid deferred loading if you want to preserve entity associations.

The following DCS example shows the three elements that represent associated `Customer`, `Employee`, and `Shipper` (empty) references:

#### **XML from DataContractSerializer**

```
<Customer i:nil="true" />
<CustomerReference
  xmlns:d2p1=
    "http://schemas.datacontract.org/2004/07/System.Data.Objects.DataClasses">
  <d2p1:EntityKey xmlns:d3p1="http://schemas.datacontract.org/2004/07/System.Data"
    i:nil="true" />
</CustomerReference>
<Employee i:nil="true" />
<EmployeeReference
  xmlns:d2p1=
    "http://schemas.datacontract.org/2004/07/System.Data.Objects.DataClasses">
  <d2p1:EntityKey xmlns:d3p1="http://schemas.datacontract.org/2004/07/System.Data"
    i:nil="true" />
</EmployeeReference>
<ShipperReference
  xmlns:d2p1=
    "http://schemas.datacontract.org/2004/07/System.Data.Objects.DataClasses">
  <d2p1:EntityKey xmlns:d3p1="http://schemas.datacontract.org/2004/07/System.Data"
    i:nil="true" />
</ShipperReference>
```

*Even if EntityKey values were available, the multiple requests to return individual entities in the foreach iterator loop would violate the "chunky not chatty" rule for service operations. The rule states that services should return data in one or a few large chunks, rather than in many small pieces because the cost of creating the cross-process connection usually is greater than that for transferring substantial amounts of data.*

### **Serialization with Eager-Loaded Associated Entities**

The modified DCS only serializes a complete object graph if you enable eager loading with the `Include()` operator, as described in the later "Eager Loading with Include() Operators" section. In

## Chapter 12: Taking Advantage of Object Services and LINQ to Entities

In this case, the serializer supplements the `EntityKey` value and replaces the `CustomerReference` with a full copy of the associated object as emphasized in the following example for the `Order.Customer` property:

### XML from `DataContractSerializer`

```
<Customer z:Id="i3">
  <EntityKey xmlns:d3p1="http://schemas.datacontract.org/2004/07/System.Data"
    z:Id="i4" xmlns=
      "http://schemas.datacontract.org/2004/07/System.Data.Objects.DataClasses">
    <d3p1:EntityContainerName>NorthwindEntities</d3p1:EntityContainerName>
    <d3p1:EntityKeyValue>
      <d3p1:EntityKeyMember>
        <d3p1:Key>CustomerID</d3p1:Key>
        <d3p1:Value xmlns:d6p1="http://www.w3.org/2001/XMLSchema"
          i:type="d6p1:string">
          COMMI
        </d3p1:Value>
      </d3p1:EntityKeyMember>
    </d3p1:EntityKeyValue>
    <d3p1:EntitySetName>Customers</d3p1:EntitySetName>
  </EntityKey>
  <Address>Av. dos Lusíadas, 23</Address>
  <City>Sao Paulo</City>
  <CompanyName>Comércio Mineiro</CompanyName>
  <ContactName>Pedro Afonso</ContactName>
  <ContactTitle>Sales Associate</ContactTitle>
  <Country>Brazil</Country>
  <CustomerID>COMMI</CustomerID>
  <Fax i:nil="true" />
  <Orders>
    <Order z:Ref="i1" />
  </Orders>
  <Phone>(11) 555-7647</Phone>
  <PostalCode>05432-043</PostalCode>
  <Region>SP</Region>
</Customer>
<CustomerReference xmlns:d2p1=
  "http://schemas.datacontract.org/2004/07/System.Data.Objects.DataClasses">
  <d2p1:EntityKey xmlns:d3p1="http://schemas.datacontract.org/2004/07/System.Data"
    z:Ref="i4" />
</CustomerReference>
```

Notice that the `<Order z:Ref="i1" />` element is a reference to — rather than a copy of — the `Order`. Specifying a reference to the `Order` entity prevents creating a cyclic relationship.

# Executing eSQL ObjectQueries

As you saw in Chapter 9, you execute an `ObjectQuery<EntityType>` against an `ObjectContext` instance to return an `ObjectQuery<EntityType>`. `ObjectContext`s are relatively weighty objects, especially if they contain a large number of business object instances that have associated entities loaded and have object change tracking enabled. Windows form apps that update the data store commonly cache `ObjectContext` instances that have multiple changes to collection members or entity property values and then apply the `ObjectContext.SaveChanges()` method to persist the changes to the data store. `ObjectContext` lifetime in Web apps commonly is limited to a Unit of Work. Martin Fowler states that the *Unit of Work* design pattern “[m]aintains a list of objects affected by a business transaction and coordinates the writing out of changes [to the data store] and the resolution of concurrency problems.” Thus, the minimum `ObjectContext` lifetime for a Web app is the duration of the simplest transaction and the maximum could be the user’s session lifetime or the server cache duration.

*There is no hard-and-fast rule or formula for optimizing ObjectContext lifetime, as you can see by executing a Web search on the words ObjectContext lifetime. Daniel Simmons, a development manager for EF, has written several blog posts (<http://blogs.msdn.com/dsimmons/>) about choosing the appropriate ObjectContext lifetime. (Search the blog for “lifetime.”)*

Most of this chapter’s examples have the option of repeating queries 1,000 times to increase elapsed time resolution for performance comparisons. The queries don’t update data, so the pattern for sample queries (without the outer `for (int i = 0; i < tries; i++)`) repetition loop is:

### C# 3.0

```
try
{
    using (NorthwindEntities ocNwind =
        new NorthwindEntities("name=NorthwindEntities"))
    {
        ObjectQuery<Order> orderQuery(eSqlText, ocNwind, MergeOption.NoTracking);
        foreach (Order order in orderQuery)
        {
            // Display the orders
        }
    }
}
catch (EntitySqlException exQry)
{
    MessageBox.Show(exQry.Message, "Entity SQL Exception");
}
catch (Exception exSys)
{
    MessageBox.Show(exSys.Message, "Other Exception");
}
```

## Chapter 12: Taking Advantage of Object Services and LINQ to Entities

---

### VB 9.0

```
Try
    Using ocNwind As New NorthwindEntities("name=NorthwindEntities")
        ObjectQuery(Of Order) orderQuery(eSqlText, ocNwind, _
            MergeOption.NoTracking)
        For Each order As Order In orderQuery
            ' Display the orders
        Next order
    End Using
Catch exQry As QueryException
    MessageBox.Show(exQry.Message, "Entity SQL Exception")
Catch exSys As Exception
    MessageBox.Show(exSys.Message, "Other Exception")
End Try
```

eSqlText is any eSQL string that returns one or more Order entities without related entities; for example:

### eSQL

```
SELECT VALUE o FROM NorthwindEntities.Orders AS o
WHERE o.ShipCountry = 'Brazil'
ORDER BY o.OrderID
DESC LIMIT(5);
```

If you attempt to execute a query with a projection that omits the VALUE keyword in order to include related entities, such as:

### eSQL

```
SELECT o, o.Customer, o.Employee, o.Shipper, o.Order_Details
FROM NorthwindEntities.Orders AS o
WHERE o.ShipCountry = 'Brazil'
ORDER BY o.OrderID
DESC LIMIT(5);
```

you receive a QueryException: “A span path can only be specified for a query with an entity or entity collection result type.” Span paths for loading associated entities are one of the topics of the next section. If your query returns an *EntityType* other than specified in the `ObjectQuery<EntityType>` or `ObjectQuery(Of EntityType)` declaration, you receive an Other Exception, such as: “Specified cast from a materialized ‘NorthwindModel.Customer’ to ‘NorthwindModel.Order’ is not valid.”

`ObjectQuery` and LINQ to Entities queries defer execution until iterated or assigned to a `List<T>`; collection, or the invocation of an `Execute(MergeOption.Option)` method or a LINQ Standard Query Operator. The `MergeOption` enumeration determines how entities from a new query merge with entities presently in the `ObjectContext` cache. The following table lists the options and their affect on the cache.

## Part V: Implementing the ADO.NET Entity Framework

MergeOption	Affect of Option on Cached Entities
AppendOnly	Only appends new entities to the cache but doesn't affect entities currently in the cache (default).
NoTracking	Doesn't add new entities to the cache or affect existing entities in the cache.
OverwriteChanges	Replaces current values in the ObjectStateEntry with values from the data store and overwrites any changes made to cached entities.
PreserveChanges	Replaces original values in the ObjectStateEntry with values from the data store to enable forcing a change in the event of a concurrency error.

The NwindObjectServicesCS.sln and NwindObjectServicesVB.sln projects in the \WROX\ADONET\Chapter12\CS and ... \VB folders lets you compare the generated T-SQL and performance of the same or equivalent eSQL queries against the EntityClient or the NorthwindEntities.Orders collection. The projects use the VisitReader() method to serialize the EntityClient's results, which aren't strongly typed, for review.

You can select the standard .NET XmlSerializer or theDataContractSerializer to review Order entities and enable object tracking to measure the performance hit for providing the ability to update entities. The lower results text box displays the serialized version of the ObjectQuery, as shown in Figure 12-1 for the XML example of the earlier "Serialization with Eager-Loaded Associated Entities" section.

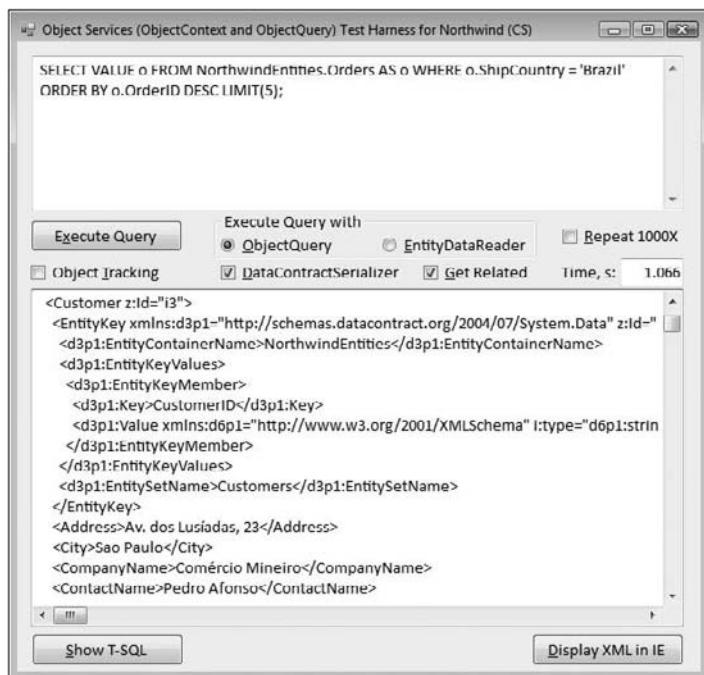


Figure 12-1

## Chapter 12: Taking Advantage of Object Services and LINQ to Entities

The following table compares execution times in milliseconds for the `EntityDataReader` to return an `SqlDataReader` and corresponding `ObjectQuery`; options include returning associated entities and enabling object tracking, which only applies to an `ObjectQuery`.

Object Type	Associated Entities	Object Tracking	Execution Time, ms.
<code>DbDataReader</code>	None	N/A	4.98
<code>DbDataReader</code>	Included	N/A	9.73
<code>ObjectQuery</code>	None	Off	9.22
<code>ObjectQuery</code>	None	On	9.71
<code>ObjectQuery</code>	Included	Off	15.00
<code>ObjectQuery</code>	Included	On	16.10

*Execution times are for the primary computer used to write this book and are the average of 5,000 executions (five executions with the Repeat 1000X check box marked) after caching the queries. Ratios of execution times for various retrieval configurations are likely valid for other computers with varying processor speeds.*

The preceding data indicates that adding the `ObjectQuery` abstraction approximately doubles the execution time for small entities. However, dramatically increasing the entity size by including associated entities with `base64Binary` encoding of the `Employee.Photo` property increases the execution time by 50 percent. This difference implies that generating the SQL statement with the Canonical Command Tree (CCT) contributes a significant part of the performance hit. The examples in the later “Compiling Entity and LINQ to Entity Queries” section proves the implication.

*The execution times shown are from a prerelease version of VS 2008 SP1 and should not be used for comparison with other O/RM tools.*

## Enabling Deferred or Eager Loading of Associated Entities

As mentioned in a note at the beginning of this chapter, LINQ to SQL lazy loads associated entities by default; EF doesn’t support lazy loading. According to the EF development team, a basic design principle is avoiding enabling by default features that might be unknown and unwanted by developers; avoiding default lazy loading adheres to that principle. The following sections describe *deferred loading* or *explicit loading*, EF’s approach that corresponds approximately to LINQ to SQL’s *lazy loading*, and *eager loading*, which is very similar to that for LINQ to SQL.

*This book uses the term *deferred loading* — rather than *explicit loading* — because loading occurs during the iteration that implements deferred execution.*

### Deferred Loading with the Load() Method

Deferred loading returns associated entities and EntitySets by testing for their existence in the cache, inspecting the Boolean `EntityNameReference.IsLoaded` or `EntityType.IsLoaded` property value and invoking the corresponding `Load()` method if the test returns false. The following snippets illustrate deferred loading code for Northwind Order entities:

#### C# 3.0

```
txtQuery = "SELECT VALUE o FROM NorthwindEntities.Orders AS o " +
           "WHERE o.ShipCountry = 'Brazil'; "
using (NorthwindEntities ocNwind = new NorthwindEntities("name=NorthwindEntities"))
{
    ObjectQuery<Order> orderQuery;
    orderQuery = new ObjectQuery<Order>(txtQuery, ocNwind,
                                         MergeOption.NoTracking)
    foreach (Order order in orderQuery)
    {
        // Defer-load each EntityRef and EntityCollection
        if (!order.CustomerReference.IsLoaded)
            order.CustomerReference.Load(MergeOption.NoTracking);
        if (!order.EmployeeReference.IsLoaded)
            order.EmployeeReference.Load(MergeOption.NoTracking);
        if (!order.ShipperReference.IsLoaded)
            order.ShipperReference.Load(MergeOption.NoTracking);
        if (!order.Order_Details.IsLoaded)
            order.Order_Details.Load(MergeOption.NoTracking);
    }
}
```

#### VB 9.0

```
txtQuery = "SELECT VALUE o FROM NorthwindEntities.Orders AS o & _
           "WHERE o.ShipCountry = 'Brazil'; "
Using ocNwind As New NorthwindEntities("name=NorthwindEntities")
    Dim orderQuery As ObjectQuery(Of Order)
    orderQuery = New ObjectQuery(Of Order)(txtQuery.Text, _
                                         ocNwind, MergeOption.NoTracking)
    For Each order As Order In orderQuery
        ' Defer-load each EntityRef and EntityCollection
        If (Not order.CustomerReference.IsLoaded) Then
            order.CustomerReference.Load(MergeOption.NoTracking)
        End If
        If (Not order.EmployeeReference.IsLoaded) Then
            order.EmployeeReference.Load(MergeOption.NoTracking)
        End If
        If (Not order.ShipperReference.IsLoaded) Then
            order.ShipperReference.Load(MergeOption.NoTracking)
        End If
        If (Not order.Order_Details.IsLoaded) Then
            order.Order_Details.Load(MergeOption.NoTracking)
        End If
    Next order
End Using
```

## Chapter 12: Taking Advantage of Object Services and LINQ to Entities

---

Deferred loading generates and executes the following simple SQL statement to return all top-level objects:

### T-SQL

```
[Extent1].[OrderID] AS [OrderID],  
[Extent1].[OrderDate] AS [OrderDate],  
[Extent1].[RequiredDate] AS [RequiredDate],  
[Extent1].[ShippedDate] AS [ShippedDate],  
[Extent1].[Freight] AS [Freight],  
[Extent1].[ShipName] AS [ShipName],  
[Extent1].[ShipAddress] AS [ShipAddress],  
[Extent1].[ShipCity] AS [ShipCity],  
[Extent1].[ShipRegion] AS [ShipRegion],  
[Extent1].[ShipPostalCode] AS [ShipPostalCode],  
[Extent1].[ShipCountry] AS [ShipCountry]  
FROM [dbo].[Orders] AS [Extent1]  
WHERE [Extent1].[ShipCountry] = 'Brazil'  
ORDER BY [Extent1].[OrderID] DESC
```

The query also generates and executes up to four additional T-SQL statements per parent entity to return the associated Customer, Employee, and Shipper entities, and the Order\_Details EntitySets. The query generates 21 T-SQL queries if object tracking is off (MergeOption.NoTracking) and 16 if object tracking is on (any other MergeOption specified). Object tracking tests for the existence of the object in the cache before executing a SQL statement to retrieve it.

## Eager Loading with Include() Operators

The `Include("EntityRef")` or `Include("EntityCollection")` operator returns entity references or collections associated with the query's parent `EntitySet`. These operators can be chained to return multiple references, multiple collections or both from a single query, as in the following example:

### C# 3.0

```
txtQuery = "SELECT VALUE o FROM NorthwindEntities.Orders AS o " +  
    "WHERE o.ShipCountry = 'Brazil'; "  
using (NorthwindEntities ocNwind = new NorthwindEntities("name=NorthwindEntities"))  
{  
    ObjectQuery<Order> orderQuery;  
    // Eager-load each EntityRef and EntityCollection  
    orderQuery = new ObjectQuery<Order>(txtQuery.Text, ocNwind,  
        MergeOption.NoTracking).Include("Order_Details").Include("Customer")  
        .Include("Employee").Include("Shipper");  
    foreach (Order order in orderQuery)  
    {  
        // Tests for loaded references and collection  
        if (!order.CustomerReference.IsLoaded)  
            throw new Exception("Customer Entity isn't loaded.");  
        if (!order.EmployeeReference.IsLoaded)  
            throw new Exception("Employee Entity isn't loaded.");
```

## Part V: Implementing the ADO.NET Entity Framework

```
        if (!order.ShipperReference.IsLoaded)
            throw new Exception("Shipper Entity isn't loaded.");
        if (!order.Order_Details.IsLoaded)
            throw new Exception("Order_Details EntitySet isn't loaded.");
    }
}
```

### VB 9.0

```
txtQuery = "SELECT VALUE o FROM NorthwindEntities.Orders AS o " & _
"WHERE o.ShipCountry = 'Brazil'; "
Using ocNwind As New NorthwindEntities("name=NorthwindEntities")
    Dim orderQuery As ObjectQuery(Of Order)
    ' Eager-load each EntityRef and EntityCollection
    orderQuery = New ObjectQuery(Of Order)(txtQuery.Text, ocNwind, _
        MergeOption.NoTracking).Include("Order_Details") _ 
        .Include("Customer").Include("Employee").Include("Shipper")
    For Each order As Order In orderQuery
        ' Tests for loaded references and collection
        If (Not order.CustomerReference.IsLoaded) Then
            Throw New Exception("Customer Entity isn't loaded.")
        End If
        If (Not order.EmployeeReference.IsLoaded) Then
            Throw New Exception("Employee Entity isn't loaded.")
        End If
        If (Not order.ShipperReference.IsLoaded) Then
            Throw New Exception("Shipper Entity isn't loaded.")
        End If
        If (Not order.Order_Details.IsLoaded) Then
            Throw New Exception("Order_Details EntitySet isn't loaded.")
        End If
    Next order
End Using
```

Following is the single, humungous T-SQL statement generated by the preceding ObjectQuery:

### T-SQL

```
SELECT
[Project1].[OrderID] AS [OrderID],
[Project1].[OrderDate] AS [OrderDate],
[Project1].[RequiredDate] AS [RequiredDate],
[Project1].[ShippedDate] AS [ShippedDate],
[Project1].[Freight] AS [Freight],
[Project1].[ShipName] AS [ShipName],
[Project1].[ShipAddress] AS [ShipAddress],
[Project1].[ShipCity] AS [ShipCity],
[Project1].[ShipRegion] AS [ShipRegion],
[Project1].[ShipPostalCode] AS [ShipPostalCode],
[Project1].[ShipCountry] AS [ShipCountry],
[Project1].[C1] AS [C1],
[Project1].[CustomerID] AS [CustomerID],
```

## Chapter 12: Taking Advantage of Object Services and LINQ to Entities

---

```
[Project1].[CompanyName] AS [CompanyName],  
[Project1].[ContactName] AS [ContactName],  
[Project1].[ContactTitle] AS [ContactTitle],  
[Project1].[Address] AS [Address],  
[Project1].[City] AS [City],  
[Project1].[Region] AS [Region],  
[Project1].[PostalCode] AS [PostalCode],  
[Project1].[Country] AS [Country],  
[Project1].[Phone] AS [Phone],  
[Project1].[Fax] AS [Fax],  
[Project1].[EmployeeID] AS [EmployeeID],  
[Project1].[LastName] AS [LastName],  
[Project1].[FirstName] AS [FirstName],  
[Project1].[Title] AS [Title],  
[Project1].[TitleOfCourtesy] AS [TitleOfCourtesy],  
[Project1].[BirthDate] AS [BirthDate],  
[Project1].[HireDate] AS [HireDate],  
[Project1].[Address1] AS [Address1],  
[Project1].[City1] AS [City1],  
[Project1].[Region1] AS [Region1],  
[Project1].[PostalCode1] AS [PostalCode1],  
[Project1].[Country1] AS [Country1],  
[Project1].[HomePhone] AS [HomePhone],  
[Project1].[Extension] AS [Extension],  
[Project1].[Photo] AS [Photo],  
[Project1].[Notes] AS [Notes],  
[Project1].[PhotoPath] AS [PhotoPath],  
[Project1].[ShipperID] AS [ShipperID],  
[Project1].[CompanyName1] AS [CompanyName1],  
[Project1].[Phone1] AS [Phone1],  
[Project1].[C2] AS [C2],  
[Project1].[OrderID1] AS [OrderID1],  
[Project1].[ProductID] AS [ProductID],  
[Project1].[UnitPrice] AS [UnitPrice],  
[Project1].[Quantity] AS [Quantity],  
[Project1].[Discount] AS [Discount]  
FROM ( SELECT  
    [Limit1].[OrderID] AS [OrderID],  
    [Limit1].[OrderDate] AS [OrderDate],  
    [Limit1].[RequiredDate] AS [RequiredDate],  
    [Limit1].[ShippedDate] AS [ShippedDate],  
    [Limit1].[Freight] AS [Freight],  
    [Limit1].[ShipName] AS [ShipName],  
    [Limit1].[ShipAddress] AS [ShipAddress],  
    [Limit1].[ShipCity] AS [ShipCity],  
    [Limit1].[ShipRegion] AS [ShipRegion],  
    [Limit1].[ShipPostalCode] AS [ShipPostalCode],  
    [Limit1].[ShipCountry] AS [ShipCountry],  
    [Extent2].[CustomerID] AS [CustomerID],  
    [Extent2].[CompanyName] AS [CompanyName],  
    [Extent2].[ContactName] AS [ContactName],  
    [Extent2].[ContactTitle] AS [ContactTitle],
```

## Part V: Implementing the ADO.NET Entity Framework

---

```
[Extent2].[Address] AS [Address],
[Extent2].[City] AS [City],
[Extent2].[Region] AS [Region],
[Extent2].[PostalCode] AS [PostalCode],
[Extent2].[Country] AS [Country],
[Extent2].[Phone] AS [Phone],
[Extent2].[Fax] AS [Fax],
[Extent3].[EmployeeID] AS [EmployeeID],
[Extent3].[LastName] AS [LastName],
[Extent3].[FirstName] AS [FirstName],
[Extent3].[Title] AS [Title],
[Extent3].[TitleOfCourtesy] AS [TitleOfCourtesy],
[Extent3].[BirthDate] AS [BirthDate],
[Extent3].[HireDate] AS [HireDate],
[Extent3].[Address] AS [Address1],
[Extent3].[City] AS [City1],
[Extent3].[Region] AS [Region1],
[Extent3].[PostalCode] AS [PostalCode1],
[Extent3].[Country] AS [Country1],
[Extent3].[HomePhone] AS [HomePhone],
[Extent3].[Extension] AS [Extension],
[Extent3].[Photo] AS [Photo],
[Extent3].[Notes] AS [Notes],
[Extent3].[PhotoPath] AS [PhotoPath],
[Extent4].[ShipperID] AS [ShipperID],
[Extent4].[CompanyName] AS [CompanyName1],
[Extent4].[Phone] AS [Phone1],
1 AS [C1],
[Extent5].[OrderID] AS [OrderID1],
[Extent5].[ProductID] AS [ProductID],
[Extent5].[UnitPrice] AS [UnitPrice],
[Extent5].[Quantity] AS [Quantity],
[Extent5].[Discount] AS [Discount],
CASE WHEN ([Extent5].[OrderID] IS NULL) THEN CAST(NULL AS int)
      ELSE 1 END AS [C2]
FROM (SELECT TOP (5) [Extent1].[OrderID] AS [OrderID], [Extent1].[CustomerID]
      AS [CustomerID], [Extent1].[EmployeeID] AS [EmployeeID],
      [Extent1].[OrderDate] AS [OrderDate],
      [Extent1].[RequiredDate] AS [RequiredDate],
      [Extent1].[ShippedDate] AS [ShippedDate],
      [Extent1].[ShipVia] AS [ShipVia], [Extent1].[Freight] AS [Freight],
      [Extent1].[ShipName] AS [ShipName],
      [Extent1].[ShipAddress] AS [ShipAddress],
      [Extent1].[ShipCity] AS [ShipCity],
      [Extent1].[ShipRegion] AS [ShipRegion],
      [Extent1].[ShipPostalCode] AS [ShipPostalCode],
      [Extent1].[ShipCountry] AS [ShipCountry]
     FROM [dbo].[Orders] AS [Extent1]
    WHERE [Extent1].[ShipCountry] = 'Brazil'
    ORDER BY [Extent1].[OrderID] DESC ) AS [Limit1]
LEFT OUTER JOIN [dbo].[Customers] AS [Extent2]
  ON [Limit1].[CustomerID] = [Extent2].[CustomerID]
LEFT OUTER JOIN [dbo].[Employees] AS [Extent3]
```

```
    ON [Limit1].[EmployeeID] = [Extent3].[EmployeeID]
    LEFT OUTER JOIN [dbo].[Shippers] AS [Extent4]
        ON [Limit1].[ShipVia] = [Extent4].[ShipperID]
    LEFT OUTER JOIN [dbo].[Order Details] AS [Extent5]
        ON [Limit1].[OrderID] = [Extent5].[OrderID]
) AS [Project1]

ORDER BY [Project1].[OrderID] ASC, [Project1].[CustomerID] ASC,
[Project1].[EmployeeID] ASC, [Project1].[ShipperID] ASC, [Project1].[C2] ASC
```

Despite the size and complexity of the preceding T-SQL query, it executes about 10 percent faster than the preceding section's 16 queries for deferred loading. As a rule eager loading with `ObjectQuery`.  
`Include()` operators will outperform deferred loading with the `EntityType.Load()` method, unless you must filter, order, or filter and order an associated `EntityCollection`.

## Ordering and Filtering Associated EntityCollections during Loading

LINQ to SQL provides the `LoadOptions.AssociateWith(LambdaExpression)` directive to enable filtering and ordering of associated entity collections at the `DataContext` level. EF offers an `EntityInstance.EntityCollection.Attach(EntityType.EntityCollection.CreateSourceQuery().LambdaExpression)` method. Unfortunately, filtering and ordering must be done for each instance of the top-level object in a `foreach` iterator loop; if the object graph has three levels or more, the number of iterators is one less than the depth of the hierarchy.

It's a common practice to work from the topmost to the bottommost entity in the hierarchy. For example, the following top-down snippet returns the `Customer-Order-Order_Detail` hierarchy with last five `Order` entities and `Order_Details` `EntitySets` for each `Customer` entity in Brazil:

### C# 3.0

```
txtQuery.Text = "SELECT VALUE c FROM NorthwindEntities.Customers " +
    "AS c WHERE c.Country = 'Brazil'; "
List<Customer> custList = new List<Customer>();
using (NorthwindEntities ocNwind = new NorthwindEntities("name=NorthwindEntities"))
{
    ObjectQuery<Customer> customerQuery =
        new ObjectQuery<Customer>(txtQuery.Text, ocNwind, mergeOpt);
    foreach (Customer customer in customerQuery)
    {
        // Get the last five orders for each customer
        customer.Orders.Attach(customer.Orders.CreateSourceQuery()
            .OrderByDescending(o => o.OrderID).Take(5));
        foreach (Order order in customer.Orders)
        {
            order.Order_Details.Attach(order.Order_Details.CreateSourceQuery());
        }
        custList.Add(customer);
    }
}
```

## Part V: Implementing the ADO.NET Entity Framework

### VB 9.0

```
txtQuery.Text = "SELECT VALUE c FROM NorthwindEntities.Customers " & _
    "AS c WHERE c.Country = 'Brazil'; " List(Of Customer) custList
Using ocNwind As New NorthwindEntities("name=NorthwindEntities")
    Dim customerQuery As ObjectQuery(Of Customer) = _
        New ObjectQuery(Of Customer)(txtQuery.Text, ocNwind, mergeOpt)
    For Each customer As Customer In customerQuery
        ' Get the last five orders for each customer
        customer.Orders.Attach(customer.Orders.CreateSourceQuery() _
            .OrderByDescending(Function(o) o.OrderID).Take(5))
        For Each order As Order In customer.Orders
            order.Order_Details.Attach(order.Order_Details.CreateSourceQuery())
        Next order
        custList.Add(customer)
    Next customer
End Using
```

The preceding approach is drastic overkill to return the same object graph (less materialized Employee and Shipper EntityRefs) as the preceding examples. The only modification of the earlier eSQL query to return the identical result *more than five times faster* is emphasized here:

### C# 3.0

```
txtQuery = "SELECT VALUE o FROM NorthwindEntities.Orders AS o " +
    "WHERE o.Customer.Country = 'Brazil';";
```

The performance-crushing culprits are the two loops required to obtain the last five Orders for each customer and each of the customer's Order\_Details Entity Set. The moral of the story is always start from the middle ground, preferably with the entity requiring a Where constraint, when returning multiple copies of three or higher tiered object graphs.

*A modified version of the first sample project (refer to Figure 12-1) in the \WROX\ADONET\Chapter12\CS\NwindOSCustomersCS and ... \VB\NwindOSCustomersVB folders runs the preceding query when you select the Entity SQL option and mark the Last Five Orders Only check box.*

Clearing the check box runs the following top-down implementation that uses the `Include("QueryPath")` directive for the two lower layers:

### C# 3.0

```
List<Customer> custList = null;
txtQuery.Text = "SELECT VALUE c FROM NorthwindEntities.Customers " +
    "AS c WHERE c.Country = 'Brazil'; "
using (NorthwindEntities ocNwind = new NorthwindEntities("name=NorthwindEntities"))
{
    // Enable eager loading of all Orders and Order_Details with Include operators
    ObjectQuery<Customer> customerQuery = new ObjectQuery<Customer>(txtQuery.Text,
        ocNwind, mergeOpt).Include("Orders").Include("Orders.Order_Details");

    // Materialize the sequence
    custList = customerQuery.ToList();
}
```

## Chapter 12: Taking Advantage of Object Services and LINQ to Entities

### VB 9.0

```
List(Of Customer) custList = Nothing
txtQuery.Text = "SELECT VALUE c FROM NorthwindEntities.Customers " & _
    "AS c WHERE c.Country = 'Brazil'; " List(Of Customer) custList
Using ocNwind As New NorthwindEntities("name=NorthwindEntities")
    ' Enable eager loading of all Orders and Order_Details with Include operators
    Dim customerQuery As ObjectQuery(Of Customer) = _
        New ObjectQuery(Of Customer)(txtQuery.Text, ocNwind, _
        mergeOpt).Include("Orders").Include("Orders.Order_Details")
    custList = customerQuery.ToList()

    ' Materialize the sequence
    custList = customerQuery.ToList()
End Using
```

This implementation executes *three times faster* than the two-loop version, while delivering *all* Orders and Order\_Details for each customer in a 705 KB DCS XML document, which includes the associated Employees and Shippers entities. The 362 KB document for the two-loop version doesn't include the extra associated entities.

## Composing Query Builder Methods to Write ObjectQueries

Chapter 9's "Composing ObjectQueries with Query Builder Methods" section describes Query Builder Methods (QBM)s as "a composable set of functions with Pascal-cased Entity SQL names that you can chain to create the equivalent of complete Entity SQL statements." Following is a copy of Chapter 9's table of QBM's and their corresponding LINQ keywords.

Query Builder Method	Entity SQL Statement	Query Builder Method	Entity SQL Statement
Distinct	DISTINCT	SelectValue	SELECT VALUE
Except	EXCEPT	Skip	SKIP
GroupBy	GROUP BY	Top	TOP and LIMIT
Intersect	INTERSECT	Union	UNION
OfType	OFTYPE	UnionAll	UNION ALL
OrderBy	ORDER BY	Where	WHERE
Select	SELECT		

The primary benefit of QBM's over eSQL is increased resistance to SQL-injection attacks against Web pages that accept user input. Their primary drawback compared with LINQ to Entities queries is that QBM's aren't strongly typed.

## Part V: Implementing the ADO.NET Entity Framework

The following snippet is the QBM implementation of the preceding eSQL query, which illustrates chaining QBMs and `Include()` operators:

### C# 3.0

```
List<Customer> custList = null;
using (NorthwindEntities ocNwind = new NorthwindEntities("name=NorthwindEntities"))
{
    // Enable eager loading of all Orders and Order_Details with Include operators
    ObjectQuery<Customer> customerQuery = ocNwind.Customers
        .Where(@"it.Country='Brazil'")
        .OrderBy("it.Country ASC")
        .Include("Orders")
        .Include("Orders.Order_Details");

    // Materialize the sequence
    custList = customerQuery.ToList();
}
```

### VB 9.0

```
List(Of Customer) custList = Nothing
Using ocNwind As New NorthwindEntities("name=NorthwindEntities")
    ' Enable eager loading of all Orders and Order_Details with Include operators
    Dim customerQuery As ObjectQuery(Of Customer) = _
        ocNwind.Customers.Where("it.Country='Brazil'"") _
            .OrderBy("it.Country ASC") _
            .Include("Orders") _
            .Include("Orders.Order_Details") _

    ' Materialize the sequence
    custList = customerQuery.ToList()
End Using
```

QBM queries, like LINQ queries, execute directly against an `ObjectContext.EntitySet` collection rather than by creating a new `ObjectQuery` instance. QBMs and `Include()` operators return `ObjectQuery<T>` types unless a LINQ method terminates the chain, in which case the query returns an `IQueryable<T>` by default.

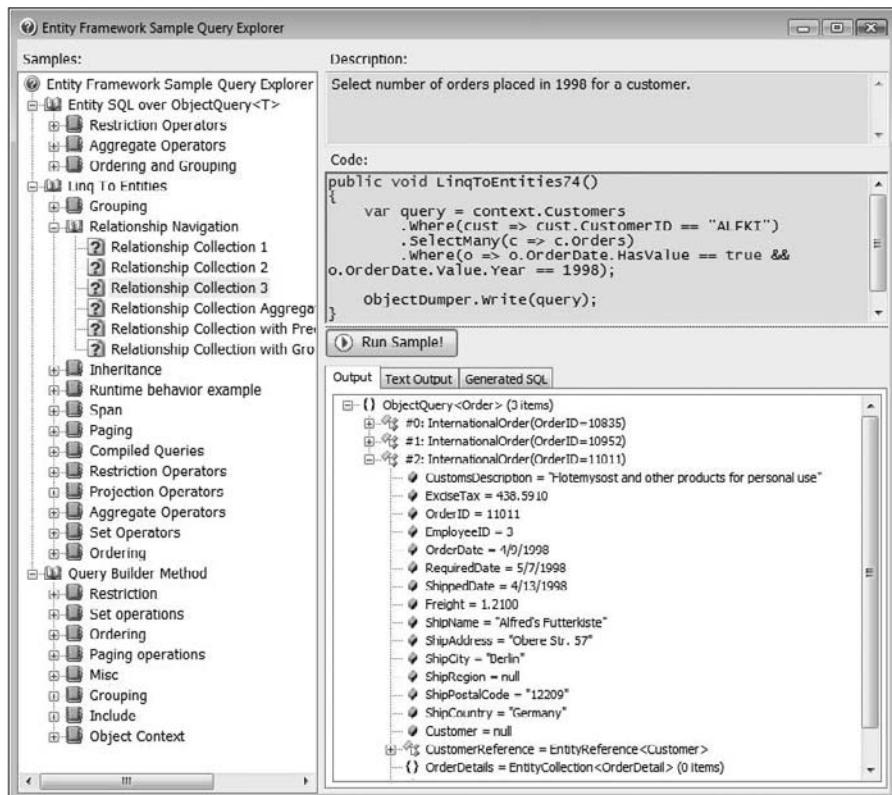
*The modified version of the first sample project in the \WROX\ADONET\Chapter12\CS\NwindOSCustomersCS and ...\VB\NwindOSCustomersVB folders runs the preceding query when you select the Query Builder Method option.*

QBM queries don't impose significant additional overhead to the corresponding eSQL query. Tests with the last query of the earlier "Ordering and Filtering Associated EntityCollections during Loading" section that uses `Include()` operators to deliver all `Orders` and `Order_Details` entities for `Customers` located in Brazil shows less than 1 percent difference in execution times.

# Writing LINQ to Entities Queries

Writing LINQ to Entities queries differs little from authoring the LINQ to SQL queries shown in Chapter 5. The Object Services layer converts the LINQ to Entities queries into a CCT that the entity-enabled data provider converts to the data store's SQL dialect. The data store returns a result set that the provider delivers as a set of `DbDataRecords`. If the `DbDataRecords` represent a single-valued projection of an entity — equivalent to the `SELECT VALUE o FROM EntitySet AS o...` statement — the returned type represents an `ObjectQuery`; otherwise, the returned sequence is a scalar or anonymous type.

The ADO.NET team has created a LINQ to Entities Query Explorer that's similar to the LINQ Query Explorer that ships as part of VS 2008. The LINQ to Entities Query Explorer executes simple sample queries and offers the option to view query results in a tree view or text representation, as well as T-SQL queries (see Figure 12-2). The EDM is based on a modified NorthwindEF database with derived `PreviousEmployees` and `InternationalOrders` types.



**Figure 12-2**

## Part V: Implementing the ADO.NET Entity Framework

---

You can download the LINQ to Entities Query Explorer's source code for VS 2008 SP1 from the ADO.NET Entity Framework Query Samples page of the MSDN Code Gallery (<http://code.msdn.microsoft.com/EFQuerySamples>).

### **Unsupported LINQ Keywords, Standard Query Operators, and Overloads**

The following list identifies LINQ keywords, Standard Query Operators (SQOs), and SQL overloads that LINQ to Entities doesn't support in expression or method call syntax:

- ❑ **Not supported:** Aggregate, DefaultIfEmpty(), ElementAt(), ElementAtOrDefault(), Last(), LastOrDefault(), Single(), SingleOrDefault(), SkipWhile(), TakeWhile()
- ❑ **Overload not supported with <IComparer> comparer argument:** OrderBy(), OrderByDescending(), ThenBy(), ThenByDescending()
- ❑ **Overload not supported with <IEqualityComparer> comparer argument:** Contains(), Distinct(), Except(), GroupBy(), GroupJoin(), Intersect(), Join(), Union()
- ❑ **Overload not supported with Func<TSource, type> selector argument:** Average(), Max(), Min(), Select(), SelectMany(), Sum(), Where()
- ❑ **Overload not supported with Func<TSource, bool> predicate argument:** Count(), LongCount()
- ❑ **Supported for EDM primitive types only:** Cast()
- ❑ **Supported for EntityTypes only:** OfType()

Most unsupported elements aren't supported because common SQL dialects don't have corresponding functions or keywords.

### **Conventional LINQ to Entities Queries**

LINQ to Entities queries use an `ObjectQuery<EntityType>` as their data source. The most commonly used LINQ to Entities queries return one or more entities or object graphs. For example, the following query returns the same result (except for data type) as the initial `ObjectQuery` example in the "Executing eSQL ObjectQueries" section near the beginning of the chapter:

#### **C# 3.0**

```
using (NorthwindEntities ocNwind = new NorthwindEntities("name=NorthwindEntities"))
{
    List<Order> orderList = null;

    ObjectQuery<Order> orders = ocNwind.Orders;
    orders.MergeOption = MergeOptions.AppendOnly;

    var orderQuery = orders.Where(o => o.Customer.Country == "Brazil")
        .OrderByDescending(o => o.OrderID)
        .Select(o => o)
```

## Chapter 12: Taking Advantage of Object Services and LINQ to Entities

```
.Take(5);

foreach (Order order in orderQuery)
{
    // Materialize the object
    orderList.Add(order);
}
}

VB 9.0

Using ocNwind As New NorthwindEntities("name=NorthwindEntities")
Dim orderList As List(Of Order) = Nothing

Dim ogOrders As ObjectQuery(Of Order) = ocNwind.Orders
orders.MergeOption = MergeOptions.AppendOnly

Dim orderQuery = ogOrders.Where(Function(o) o.Customer.Country = "Brazil") _
                    .OrderByDescending(Function(o) o.OrderID) _
                    .Select(Function(o) o) _
                    .Take(5)

For Each ogOrder As Order In orderQuery
    ' Materialize the object
    orderList.Add(ogOrder)
Next order
End Using
```

The data type returned by LINQ to Entities queries that return EDM EntityTypes is `IQueryable<EntityType>` by default.

*The examples of this and the following sections use LINQ method-call syntax because it's similar to QBM syntax and is consistent with `Include()` operator usage.*

## Using the `Include()` Operator with LINQ to Entities Queries

Returning partial or complete object graphs with associated entities requires executing the `Include()` operator(s) before the LINQ SQOs, as shown in the following query, which returns the same result as that of the earlier “Eager Loading with `Include()` Operators” section:

### C# 3.0

```
using (NorthwindEntities ocNwind = new NorthwindEntities("name=NorthwindEntities"))
{
    List<Order> orderList = null;

    ObjectQuery<Order> orders = ocNwind.Orders;
    orders.MergeOption = MergeOptions.AppendOnly;

    var orderQuery = orders.Include("Order_Details")  
                      .Include("Customer")  
                      .Include("Employee")
```

## Part V: Implementing the ADO.NET Entity Framework

```
.Include("Shipper")
.Where(o => o.Customer.Country == "Brazil")
.OrderByDescending(o => o.OrderID)
.Select(o => o)
.Take(5);

foreach (Order order in orderQuery)
{
    // Materialize the object
    orderList.Add(order);
}
}
```

### VB 9.0

```
Using ocNwind As New NorthwindEntities("name=NorthwindEntities")
    Dim orderList As List(Of Order) = Nothing

    Dim ogOrders As ObjectQuery(Of Order) = ocNwind.Orders
    orders.MergeOption = MergeOptions.AppendOnly

    Dim orderQuery = ogOrders.Include("Order_Details") _
        .Include("Customer") _
        .Include("Employee") _
        .Include("Shipper") _
        .Where(Function(o) o.Customer.Country = "Brazil") _
        .OrderByDescending(Function(o) o.OrderID) _
        .Select(Function(o) o) _
        .Take(5)

    For Each ogOrder As Order In orderQuery
        ' Materialize the object
        orderList.Add(ogOrder)
    Next ogOrder
End Using
```

The LINQ expression syntax has more than the ordinary differences from the chained method call. The type change from `ObjectQuery<Order>` to `IQueryable<Order>` that occurs after the four SQOs execute requires segmenting the `Include()` operators from the LINQ expression, as shown here:

### C# 3.0

```
orders = orders.Include("Order_Details")
    .Include("Customer")
    .Include("Employee")
    .Include("Shipper");

orderQuery = (from o in orders
    where o.Customer.Country == "Brazil"
    orderby o.OrderID descending
    select o).Take(5);
```

## Chapter 12: Taking Advantage of Object Services and LINQ to Entities

---

### VB 9.0

```
oqOrders = oqOrders.Include("Order_Details")
    .Include("Customer")
    .Include("Employee")
    .Include("Shipper")

orderQuery = (From o In oqOrders _
    Where o.Customer.Country = "Brazil" _
    Order By o.OrderID Descending _
    Select o).Take(5)
```

Alternatively, you can use this slightly less legible syntax:

### C# 3.0

```
orderQuery = (from o in orders.Include("Order_Details")
    .Include("Customer")
    .Include("Employee")
    .Include("Shipper")
    where o.Customer.Country == "Brazil"
    orderby o.OrderID descending
    select o).Take(5);
```

### VB 9.0

```
orderQuery = (From o In oqOrders.Include("Order_Details" _
    .Include("Customer") _
    .Include("Employee") _
    .Include("Shipper") _
    Where o.Customer.Country = "Brazil" _
    Order By o.OrderID Descending _
    select o).Take(5);
```

## **Compiling LINQ to Entity Queries**

EF and LINQ to SQL have the capability to compile and cache LINQ queries. Compiling the query takes longer than ordinary execution, but successive executions usually run much faster. The `CompiledQuery` class exposes a `Compile()` method that has the following signature plus three overloads that support passing one (shown as `Arg1` below), two or three parameter values:

### C# 3.0

```
public static Func<TArg0, TResult> Compile<TArg0, TResult>
    (Expression<Func<TArg0, TResult>> query) where TArg0: ObjectContext;
public static Func<TArg0, TArg1, TResult> Compile<TArg0, TArg1, TResult>
    (Expression<Func<TArg0, TArg1, TResult>> query) where TArg0: ObjectContext;
```

### VB 9.0

```
Public Shared Function Compile(Of TArg0 As ObjectContext, TResult)
    (ByVal query As Expression(Of Func(Of TArg0, TResult)))
        As Func(Of TArg0, TResult)
Public Shared Function Compile(Of TArg0 As ObjectContext, TArg1, TResult)
    (ByVal query As Expression(Of Func(Of TArg0, TArg1, TResult)))
        As Func(Of TArg0, TArg1, TResult)
```

## Part V: Implementing the ADO.NET Entity Framework

---

You call the `Compile()` method with parameters shown here:

```
var compiledQuery = <ObjectContextType, DataTypeName, IQueryable<EntityType>>
    (LambdaFunction)
```

The `LambdaFunction` passes the `ObjectContext` instance and parameter value to the query. You invoke the method with a call like this to supply the instance and parameter value:

```
var query = compiledQuery.Invoke(ObjectContextInstance, ParamValue)
```

Following are examples of classes for compiling the preceding section's two Orders queries:

### C# 3.0

```
static readonly Func<NorthwindEntities, String, IQueryable<Order>> compQuery =
    CompiledQuery.Compile<NorthwindEntities, String, IQueryable<Order>>
        ((ctxComp, country) => ctxComp.Orders.Where(o => o.Customer.Country == country)
            .OrderByDescending(o => o.OrderID)
            .Select(o => o)
            .Take(5));
```

### VB 9.0

```
ReadOnly compQuery As Func(Of NorthwindEntities, String, IQueryable(Of Order)) = _
    CompiledQuery.Compile(Of NorthwindEntities, String, _
    IQueryable(Of Order))(Function(ctxComp, country)
        ctxComp.Orders.Where(Function(o) o.Customer.Country = country) _
            .OrderByDescending(Function(o) o.OrderID) _
            .Select(Function(o) o) _
            .Take(5))
```

Examples with the `Include()` operator:

### C# 3.0

```
static readonly Func<NorthwindEntities, String, IQueryable<Order>> compQuery =
    CompiledQuery.Compile<NorthwindEntities, String, IQueryable<Order>>
        ((ctxComp, country) => ctxComp.Orders.Include("Order_Details")
            .Include("Customer")
            .Include("Employee")
            .Include("Shipper")
            .Where(o => o.Customer.Country == country)
            .OrderByDescending(o => o.OrderID)
            .Select(o => o)
            .Take(5));
```

### VB 9.0

```
ReadOnly compQuery As Func(Of NorthwindEntities, String, IQueryable(Of Order)) = _
    CompiledQuery.Compile(Of NorthwindEntities, String, _
    IQueryable(Of Order))(Function(ctxComp, country) ctxComp.Orders _
        .Include("Order_Details")
        .Include("Customer")
        .Include("Employee")
        .Include("Shipper")
```

## Chapter 12: Taking Advantage of Object Services and LINQ to Entities

```
.Where(Function(o) o.Customer.Country = country) _  
.OrderByDescending(Function(o) o.OrderID) _  
.Select(Function(o) o) _  
.Take(5))
```

The first time you invoke `compQuery`, the compiler compiles and caches the query. The cached version is valid for the lifetime of your `ObjectContext` instance.

*The `sp_executesql` stored procedure executes compiled LINQ queries; uncompiled LINQ queries send T-SQL batches.*

### Comparing the Performance of LINQ to Entities Queries

The test harnesses for LINQ to Entities queries in the \WROX\ADONET\Chatper12\CS\NwindLinqOrdersCS and ... \VB\NwindLinqOrdersVB folders execute the queries against the Orders EntitySet in the preceding sections and share `DbDataReader` code with the other test harness versions. The following table compares the execution time of uncompiled and compiled LINQ to Entities queries with corresponding queries that use eSQL to deliver `ObjectQuery` objects:

Object/Query Type	Associated Entities	Object Tracking	Execution Time 1, ms.	Execution Time 2, ms.	Compiled Time, ms.
ObjectQuery	None	Off	9.65	1.37	N/A
ObjectQuery	None	On	10.21	0.93	N/A
ObjectQuery	Included	Off	17.90	9.11	N/A
ObjectQuery	Included	On	17.89	6.22	N/A
LINQ to Entities	None	Off	16.34	7.70	2.12
LINQ to Entities	None	On	19.35	10.07	2.27
LINQ to Entities	Included	Off	55.46	46.41	9.08
LINQ to Entities	Included	On	61.91	52.69	9.19

*Execution Time 1 includes time to instantiate a new `ObjectContext` for each query execution, which caches the `ObjectQuery` and parts of a LINQ query. Execution Time 2 use the same `ObjectContext` instance for all 1,000 executions for more appropriate comparison with Compiled time.*

It's clear that substituting LINQ to Entities for an eSQL `ObjectQuery` exacts a significant performance hit when creating a complex object graph from the resultset of the very large T-SQL statement listed in the earlier "Eager Loading with `Include()` Operators" section. As expected, the LINQ to Entities query with `Include()` operators generates a T-SQL query that's identical to the `ObjectQuery`'s. The most arresting results are the dramatic decrease in query execution time by compiling parameterized queries.

# Parameterizing Object Queries

The `ObjectQuery.Parameters` collection enables adding `ObjectParameter(ParamName, ParamValue)` members to deliver values to @ParamName placeholders of eSQL text and query builder method queries. The following examples assume prior declaration of a `ctxNwind` `ObjectContext` and `mergeOpt` `MergeOption`:

### C# 3.0

```
string paramSql = "SELECT VALUE o FROM NorthwindEntities.Orders AS o " +
"WHERE o.ShipCountry = @country ORDER BY o.OrderID DESC LIMIT(5); ";

objectQuery = new ObjectQuery<Order>(paramSql, ctxNwind, mergeOpt)
objectQuery.Parameters.Add(new ObjectParameter("country", "Brazil"));
ObjectQuery<Order> objectQuery = ctxNwind.Orders
    .Where(@"it.ShipCountry=@country")
    .OrderBy("it.OrderID DESC")
    .Top("5");
objectQuery.Parameters.Add(new ObjectParameter("country", "Brazil"));
```

### VB 9.0

```
Dim paramSql As String = "SELECT VALUE o FROM NorthwindEntities.Orders AS o " &
"WHERE o.ShipCountry = @country ORDER BY o.OrderID DESC LIMIT(5); "

objectQuery = New ObjectQuery(Of Order)(paramSql, ctxNwind, mergeOpt)
objectQuery.Parameters.Add(New ObjectParameter("country", "Brazil"));
Dim objectQuery As ObjectQuery(Of Order) = _
    ctxNwind.Orders.Where("it.ShipCountry=@country") _
        .OrderBy("it.OrderID DESC") _
        .Top("5")
objectQuery.Parameters.Add(New ObjectParameter("country", "Brazil"));
```

Entity SQL isn't likely to be subject to SQL injection attacks because the CCT translates only valid eSQL or query builder input to the data provider's SQL dialect. However, DBAs aren't likely to take a chance on sending parameter values as clear text, so parameterizing them is worth the extra line of code each.

*The `sp_executesql` stored procedure executes parameterized object queries; queries that use nonparameterized variable value or eSQL changes send T-SQL batches.*

# Summary

The hierarchical `DbDataRecord` objects delivered by eSQL queries against the `EntityClient` probably will interest some .NET developers because this simple configuration can provide applications with a "plug-in" feature for RDBMSs with EF-enabled data providers that use different SQL dialects. eSQL's lack of DML `INSERT`, `UPDATE`, and `DELETE` commands in v1 probably will severely limit its deployment in this scenario.

## Chapter 12: Taking Advantage of Object Services and LINQ to Entities

---

Mainstream applications undoubtedly will use the Object Services API and most will use strongly typed LINQ to Entities queries to return complete entities or entity collections. EF doesn't implement lazy loading by default; in fact, EF doesn't implement on-demand loading at all. Instead, you can choose between eager loading of all related entities with the `Include()` operator or deferred (until iteration) loading with `If Not RelatedEntity.IsLoaded Then Load(RelatedEntity)` structures in the iterator loop. If you want to load a specific set of related entities associated with your target entity in a one:many relationship, make the entity requiring the `WHERE` and, if applicable, `ORDER BY` clause the target (principal) entity; for example, `Order` in a `Customer-Order-Order_Detail` hierarchy. Otherwise, you must invoke the `CreateSourceQuery()` method with a LINQ to Entities query for the desired associated entities as its argument.

.NET Framework 3.5 SP1 made a minor modification to the `DataContractSerializer`, and VS 2008 SP1 added support for serializing object graphs with cyclic references created by one:many relationships. This new feature relies on applying the `DataContract` attribute with its `IsReference=true` attribute to each `EntityType` class and `DataMember` attribute to each class property in the generated `ModelName.Designer.cs` or `.vb` file. The effect of this update is to make `EntityTypes` transportable across machine boundaries to begin implementing SOA with the EDM and WCF. However, read-write Web services won't be available without third-party modifications until the EF developers deliver a serializable `ObjectStateEntries` collection or its equivalent to persist updates by the service's client in the middle-tier's data store.

Query builder methods add a LINQ-like flavor to object queries, but LINQ to Entities delivers the real LINQ experience, less a few unsupported SQOs and SQO overloads. LINQ to Entities queries support prefixing `Include()` operators to LINQ query expressions or method call syntax and compiling as well as caching queries for improved performance of repeated execution of the same query or a query whose parameter values vary. Parameterizing eSQL, query builder method, and LINQ queries relies on the `ObjectQuery.Parameters` collection of `ObjectParameter` members, which are analogous to `DbCommand.Parameters` and `DbParameter` objects.



# 13

## Updating Entities and Complex Types

The preceding three chapters about Entity Framework (EF) and the Entity Data Model (EDM) deal primarily with mapping from the store to the conceptual schema and querying the EDM with Entity SQL (eSQL) and LINQ to Entities. This chapter primarily addresses create, update, and delete (CUD) operations on entity instances and collections, including optimistic concurrency management. It also covers the process of detaching entities from an existing `ObjectContext` instance and attaching them to new `ObjectContext` instances, which is an important factor in the use of the EDM as a data source in *n*-tier, service-oriented applications.

The chapter also deals with mapping and updating complex types, which also are known as *value objects*. *Complex types* are data structures, which resemble entities that have multiple properties but don't have a unique identity. The most common example of a complex type is `Address`, which might have `StreetAddress`, `City`, `Region`, `PostalCode`, and `Country` properties.

*Sample projects for this chapter are in the \WROX\ADONET\Chapter13\CS and . . . \VB folders.*

### Understanding the ObjectContext's ObjectStateManager and Its Children

Chapter 9's "ObjectStateManager" section introduced you to EF's `ObjectContext`'s `ObjectStateManager`, which is responsible for managing entity instances' identity, providing change tracking, and supplying a change set that you can use for resolving concurrency conflicts. Chapter 9's primary context is an overview of the Entity Data Model (EDM), Entity SQL (eSQL) and Object Services, as well as `EntityType`s, `EntitySet`s, and LINQ to Entities queries. This chapter deals primarily with creating, updating, and deleting entities; these operations rely heavily on the `ObjectStateManager`.

## Part V: Implementing the ADO.NET Entity Framework

---

By default, the `ObjectStateManager` assigns a unique `EntityKey` to each `EntityObject` instance in the `ObjectContext`'s `EntitySets` and adds it to a local object cache. For example, the `NorthwindModel` EDM with the eight original tables, which is the basis for most of the sample code in this book, generates 7,372 entity `EntityObject` instances. The following table lists and describes the `ObjectStateManager`'s most commonly used methods and its event.

Name	Description
<b>Methods</b>	
<code>GetObjectStateEntries(EntityState.State)</code>	Retrieves a collection of <code>ObjectStateEntry</code> objects for the specified <code>EntityState</code>
<code>GetObjectStateEntry(EntityObject.EntityKey)</code>	Retrieves the corresponding <code>ObjectStateEntry</code> for the specified <code>EntityKey</code>
<code>TryGetObjectStateEntry(EntityObject.EntityKey)</code>	Attempts to return the corresponding <code>ObjectStateEntry</code> for the specified <code>EntityKey</code>
<b>Event</b>	
<code>ObjectStateManagerChanged</code>	Fires when adding or removing entity instances

Each instance of an `EntityObject` has  `EntityState` and  `EntityKey` properties.  `EntityState` is an enumeration with four valid entity states with respect to the current `ObjectContext`: `Unchanged`, `Modified`, `Added`, or `Deleted`, which reflects the current state of the instance. A fifth state, `Detached`, isn't valid for an instance attached to an `ObjectContext`.

The following table lists and describes the  `EntityKey`'s methods and event.

Name	Description
<b>Methods</b>	
<code>Equals (Overloaded)</code>	Returns <code>true</code> if the instance is equal to a specified object; <code>false</code> otherwise
<code>GetEntitySet</code>	Gets the entity key's entity set from the metadata workspace
<code>GetHashCode (Overridden)</code>	A hash function for the current <code> EntityKey</code> object, which can be used with hashing algorithms and data structures such as a hash table
<code>OnDeserialized</code>	Helper method for deserializing an <code> EntityKey</code>
<code>OnDeserializing</code>	Helper method for deserializing an <code> EntityKey</code>

Name	Description
<b>Properties</b>	
EntityContainerName	Gets the EntityContainer's name
EntityKeyValues	Gets the EntityKey's key values
EntitySetName	Gets the EntitySet's name
IsTemporary	Returns <code>true</code> if the EntityKey is temporary, <code>false</code> otherwise

`EntityKeyValues` are a collection of `KeyValuePair<TKey, TValue>` generic structures, which represent the foreign key values of an association. Two or more `KeyValuePair`s represent composite primary keys of the associated entities persistence table.

## Updating or Deleting Entities

Entity SQL (eSQL) v1 doesn't support SQL's Data Manipulation Language (DML), so updating and deleting entities requires instantiating them if they aren't present in the `ObjectContext`'s local cache. Invoking the `EDMNameEntities` class's default constructor caches the `EntitySets` for the `DataContext`'s lifetime, as shown here:

### C# 3.0

```
using (NorthwindEntities ctxNwind = new NorthwindEntities())
{
    ...
}
```

### VB 9.0

```
Using ctxNwind As New NorthwindEntities()
    ...
End Using
```

Use code like the following to update an entire `EntitySet`, such as `Products`, by increasing the `UnitPrice` of all items by 10 percent:

### C# 3.0

```
using (NorthwindEntities ctxNwind = new NorthwindEntities())
{
    foreach (Product p in ctxNwind.Products)
        p.UnitPrice *= 1.1M;
}
```

### VB 9.0

```
Using ctxNwind As New NorthwindEntities()
    For Each p As Product In ctxNwind.Products
        p.UnitPrice *= 1.1;
    End For
End Using
```

## Part V: Implementing the ADO.NET Entity Framework

---

C# and VB example code that's similar to the procedures in this and the following sections is located in the \WROX\ADONET\Chapter13\CS\NwindUpdatesCS and .\VB\NwindUpdatesVB folders.

### Persisting Changes to the Data Store

To persist the changes to the underlying tables, invoke the `ObjectContext.SaveChanges(acceptChangesDuringSave)` method:

#### C# 3.0

```
using (NorthwindEntities ctxNwind = new NorthwindEntities())
{
    foreach (Product p in ctxNwind.Products)
        p.UnitPrice *= 1.1M;
    ctxNwind.SaveChanges(true);
}
```

#### VB 9.0

```
Using ctxNwind As New NorthwindEntities()
    For Each p as Product In ctxNwind.Products
        p.UnitPrice *= 1.1;
    End For
    ctxNwind.SaveChanges(True);
End Using
```

The `acceptChangesDuringSave`'s argument defaults to `true`; setting the argument to `false` requires a subsequent `ObjectContextName.AcceptAllChanges()` method call to synchronize the local cache with the persisted values.

### Logging Entities' State

You can execute code between the `SaveChanges()` and `AcceptAllChanges()` invocations that logs the number or identity of entities in the four allowed `EntityStates`: `Unchanged`, `Modified`, `Added`, or `Deleted` by calling the `ObjectStateManager.GetObjectStateEntries()` method as shown in the following fragment:

#### C# 3.0

```
StringBuilder log = new StringBuilder();
ObjectStateManager stateMgr = context.ObjectStateManager;
IEnumerable<ObjectStateEntry> unchangedState =
    stateMgr.GetObjectStateEntries(EntityState.Unchanged);
log.Append(unchangedState.Count().ToString() + " Unchanged entities\r\n");
```

#### VB 9.0

```
Dim log As New StringBuilder();
Dim stateMgr As ObjectStateManager = context.ObjectStateManager
Dim unchangedState As IEnumerable(Of ObjectStateEntry) =
    stateMgr.GetObjectStateEntries(EntityState.Unchanged)
log.Append(unchangedState.Count().ToString() + " Unchanged entities" + vbCrLf)
```

### ***Deleting Selected Entities***

Deleting entities is as simple as updating them. To delete a specific entity instance or set, write an eSQL ObjectQuery or LINQ to Entities expression or method to return instances of entities to delete, and invoke the `DataContext.DeleteObject(Entity)` method in the query's iterator loop:

#### **C# 3.0**

```
using (NorthwindEntities ctxNwind = new NorthwindEntities())
{
    var custs = from c in ctxNwind.Customers
                where c.CustomerID.StartsWith("ZZ")
                select c;
    foreach (Customer c in custs)
        ctxNwind.DeleteObject(c);
    ctxNwind.SaveChanges(true);
}
```

#### **VB 9.0**

```
Using ctxNwind As New NorthwindEntities()
    Dim Custs = From c In ctxNwind.Customers _
                 Where c.CustomerID.StartsWith("ZZ") _
                 Select c;
    For Each c As Customer In ctxNwind.Customers
        ctxNwind.DeleteObject(c)
    End For
    ctxNwind.SaveChanges(True);
End Using
```

There's little or no incentive to use an eSQL query rather than a LINQ to Entities query for updates or deletions. The ADO.NET Entity Framework (EF) emulates LINQ to SQL's data update approach with only minor syntax changes.

### ***Updating Associated EntitySets or EntityObjects***

Chapter 9 mentioned the `MergeOption` enumeration in the context of eSQL ObjectQueries. `MergeOption` controls the updatability and original/current values of an `EntityObject`'s associated `EntitySets` or `EntityObjects` when used in the context of an `DataContext.Load(MergeOption.ChosenOption)` statement. `ChosenOption` must be one of the four `MergeOption` members described the following table.

## Part V: Implementing the ADO.NET Entity Framework

MergeOption	Description
AppendOnly	Will only append new entities and will not modify existing entities that were previously fetched. This is the default behavior.
NoTracking	Will not modify ObjectStateManager.
OverwriteChanges	The current values in the ObjectStateEntry will be replaced with the values from the store. This will overwrite the changes that have been made locally with data from the server.
PreserveChanges	The original values will be replaced without modifying the current values. This is useful for forcing the local values to save successfully after an optimistic concurrency exception.

If you select the NoTracking option to improve performance, you must attach the associated EntityObject(s) before you can delete or modify them in conjunction with deletion or modification of the parent object.

### ***Deleting Entities with Dependent Associated Entities***

If the EntityObject instance your deleting has a one:many association with a dependent EntitySet and the database doesn't implement cascading deletions on the corresponding relationship, you must instantiate and delete the associated entities before you delete the parent entity.

*Many:one associations, such as Order.Customer seldom are dependent; that is a Customer entity doesn't require an Order entity and, therefore, isn't dependent on Order. On the other hand, OrderLineItem entities depend on the existence of a corresponding Order entity.*

If you enable cascading deletions in the database, be sure that the `<End Role="ParentEntity">` `</End>` node of the Conceptual Model's `<Association>` element contains an `<OnDelete Action="Cascade">` element.

### ***Transacting Updates and Deletions***

Invoking the `ObjectContext.SaveChanges()` method begins a transaction which automatically rolls back all changes persisted to the database if an exception occurs before iteration completes; otherwise, the transaction commits. You might be tempted to apply the method after each entity update or deletion, rather than after iteration completes, especially when you're updating or deleting massive numbers of entities.

If you try to invoke `ObjectContext.SaveChanges()` before all data has been processed, you incur a `System.Data.SqlClient.SqlException`: "New transaction is not allowed because there are other threads running in the session." The exception occurs because SQL Server 2005+ doesn't permit starting a new transaction on a connection that has a `SqlDataReader` open, even with Multiple Active Record Sets (MARS) enabled by the connection string. (EF's default connection string enables MARS.)

# Adding Entities

Adding a new entity requires creating a new `EntityObject` instance, setting at least its required property values, and invoking the `ObjectContext.AddToEntitySetName(EntityType)` method, as in the following example:

### C# 3.0

```
using (NorthwindEntities ctxNwind = new NorthwindEntities())
{
    Customer cust = new Customer();
    cust.CustomerID = "Bogus";
    cust.CompanyName = "Bogus Software, Inc.";
    ...
    cust.Fax = "(510) 555-1213"
    ctxNwind.AddToCustomers(cust);
    recsInserted += ctxNwind.SaveChanges(true);
}
```

### VB 9.0

```
Using ctxNwind As New NorthwindEntities()
    ctxNwind.SaveChanges(True)
    Customer Cust = new Customer()
    Cust.CustomerID = "Bogus";
    Cust.CompanyName = "Bogus Software, Inc.";
    ...
    Cust.Fax = "(510) 555-1213"
    ctxNwind.AddToCustomers(Cust);
    recsInserted += ctxNwind.SaveChanges(true);
End Using
```

The “New transaction is not allowed . . .” exception mentioned in the preceding *Transacting Updates and Deletions* section doesn’t occur when you invoke the `SaveChanges()` method before adding all new entities to the `ObjectContext` because no iterator is holding an `SqlDataReader` open.

# Refreshing Stale Entities

If you don’t enable Object Services, you must send out-of-band `INSERT`, `UPDATE`, or `DELETE` SQL commands or execute stored procedures in the data provider’s SQL dialect. Out-of-band updates require refreshing the resultset of any eSQL `SELECT` queries that might contain modified source data. Additionally, your cached entities might become stale if you hold the `ObjectContext` open for a long time in a Windows form app or a middle-tier Web service.

In either case, you can invoke the `ObjectContext.Refresh({EntitySet|entity instance}, RefreshMode.StoreWins)` method to synchronize an `EntitySet` or entity instance to the persisted values. Substituting `RefreshMode.ClientWins` doesn’t synchronize cached instances but updates the persistence store with locally cached values upon the next `SaveChanges()` invocation.

Refreshing individual `EntitySets` is faster than regenerating the `ObjectContext` from scratch because refreshing avoids loading and initializing metadata from the schema and mapping files, and omits view generation.

# Validating Data Additions and Updates

The `ObjectContext.SaveChanges()` method fires `SavingChanges()` events before and `SavedChanges()` events after the data store persists updates, additions, or deletions. Wiring up the changes with a C# `this.SavingChanges += NwindSavingChanges;` statement gives you access to current `ObjectContext` instance and its `ObjectStateManager`, which provides access to `ObjectStateEntries` collections classified by `EntityState`.

Iterating by `EntityObject` type lets you perform common operations, such as verifying presence of required property values, and adding audit field values — typically `UpdatedOn` and `UpdatedBy` or `CreatedOn` and `CreatedBy` values, as in the following examples:

### C# 3.0

```
namespace NorthwindModel
{
    public partial class NorthwindEntities
    {
        private void NwindSavingChanges(object sender, EventArgs e)
        {
            var modEntities = this.ObjectStateManager.
                GetObjectStateEntries(EntityState.Modified);
            if (modEntities.Count() > 0)
            {
                // Updating entities
                MainForm.ActiveForm.Controls["txtLog"].Text += "Validating " +
                    modEntities.Count().ToString() + " updated entities.\r\n";
                foreach (var ent in modEntities)
                {
                    if (ent.Entity is Order_Detail)
                    {
                        Order_Detail entity = (Order_Detail)ent.Entity;
                        // Add UpdatedOn and UpdatedBy
                    }
                }
            }

            var newEntities = this.ObjectStateManager.
                GetObjectStateEntries(EntityState.Added);
            if (newEntities.Count() > 0)
            {
                // Adding entities
                MainForm.ActiveForm.Controls["txtLog"].Text += "Validating " +
                    newEntities.Count().ToString() + " added entities.\r\n";
                foreach (var ent in newEntities)
                {
                    if (ent.Entity is Customer)
                    {
                        Customer entity = (Customer)ent.Entity;
                        // Add CreatedOn and CreatedBy
                    }
                }
            }
        }
    }
}
```

## Chapter 13: Updating Entities and Complex Types

---

```
var delEntities = this.ObjectStateManager.  
    GetObjectStateEntries(EntityState.Deleted);  
if (delEntities.Count() > 0)  
{  
    // Deleting entities  
    MainForm.ActiveForm.Controls["txtLog"].Text += "Validating " +  
        delEntities.Count().ToString() + " deleted entities.\r\n"  
    foreach (var ent in delEntities)  
    {  
        if (ent.Entity is Customer)  
        {  
            // Tests but no validation  
        }  
    }  
    Application.DoEvents();  
}  
}  
  
// Individual handlers for validating entity properties  
public partial class Customer  
{  
    partial void OnCustomerIDChanging(string value)  
    {  
        string custID = value;  
        // Validation tests  
    }  
}  
  
public partial class Order_Detail  
{  
    partial void OnQuantityChanging(short value)  
    {  
        short Quantity = value;  
        // Validation tests  
    }  
}
```

The Entity Object partial classes include `OnPropertyNameChanging` and `OnPropertyNameChanged` partial methods as event delegates. The two partial class and method examples at the end of the preceding example show how to validate individual property values.

## Part V: Implementing the ADO.NET Entity Framework

---

A Handles Me.SavingChanges clause wires the following VB version of the event handler to the appropriate event:

### VB 9.0

```
Namespace NorthwindModel
    Partial Public Class NorthwindEntities
        'Handler for ObjectContext.SavingChanges event
        Private Sub NwindSavingChanges(ByVal sender As Object, _
            ByVal e As EventArgs) Handles Me.SavingChanges
            Dim modEntities = Me.ObjectStateManager. _
                GetObjectStateEntries(EntityState.Modified)
            If modEntities.Count > 0 Then
                ' Updating entities
                MainForm.Controls("txtLog").Text += "Validating " + _
                    modEntities.Count.ToString + " updated entities." + vbCrLf
                For Each ent In modEntities
                    If TypeOf ent.Entity Is Order_Detail Then
                        Dim entity As Order_Detail = _
                            CType(ent.Entity, Order_Detail)
                        ' Add UpdatedOn and UpdatedBy
                        ' Stop 'test
                    End If
                Next
            End If

            Dim newEntities = Me.ObjectStateManager. _
                GetObjectStateEntries(EntityState.Added)
            If newEntities.Count > 0 Then
                ' Adding entities
                MainForm.Controls("txtLog").Text += "Validating " + _
                    newEntities.Count.ToString + " added entities." + vbCrLf
                For Each ent In newEntities
                    If TypeOf ent.Entity Is Customer Then
                        Dim entity As Customer = CType(ent.Entity, Customer)
                        ' Add CreatedOn and CreatedBy
                        ' Stop 'test
                    End If
                Next
            End If

            Dim delEntities = Me.ObjectStateManager. _
                GetObjectStateEntries(EntityState.Deleted)
            If delEntities.Count > 0 Then
                ' Deleting entities
                MainForm.Controls("txtLog").Text += "Validating " + _
                    delEntities.Count.ToString + " deleted entities." + vbCrLf
                For Each ent In delEntities
                    If TypeOf ent.Entity Is Customer Then
                        'Stop 'test
                    End If
                Next
            End If
            Application.DoEvents()
        End Sub
    End Class
End Namespace
```

```
End Class

' Individual handlers for validating entity properties
Partial Public Class Customer
    Private Sub OnCustomerIDChanging(ByVal value As String)
        Dim custID As String = value
        ' Stop 'test
    End Sub
End Class

Partial Public Class Order_Detail
    Private Sub OnQuantityChanging(ByVal value As Short)
        Dim Quantity As Short = value
        ' Stop 'test
    End Sub
End Class
End Namespace
```

## Optimizing the ObjectContext Lifetime

Microsoft positions EF as a replacement for ADO.NET DataSets, but EF doesn't support occasionally connected user scenarios with disconnected operations, which is one of the DataSet's primary features. When users make changes to a DataSet while offline (disconnected from the data source), the changes are stored in XML UpdateGrams within the DataSet's appropriate table group. When the user reconnects and executes the `DataSet.ApplyChanges()` method, updates, or deletions that don't encounter exceptions, update the underlying tables. EF's `DataContext` is associated with a `DbConnection` object, which the `DataContext` opens to retrieve or update data and closes immediately thereafter. Disposing the `DataContext` closes the connection and disposes all attached objects.

Deciding the optimum `ObjectContext` lifetime for a specific project or category of projects isn't easy, because the EF team's members haven't published official guidance for the topic. In addition, the performance hit and database load for creating an `ObjectContext` from scratch depends largely on the number of `EntityTypes` in your EDM and the size of their `EntitySets`. Daniel Simmons, EF's development lead, suggested "using a single context (or one per thread if your app is multi-threaded)" in a March 2008 blog post. In a February 2008 post ([blogs.msdn.com/dsimmons/archive/2008/02/17/context-lifetimes-dispose-or-reuse.aspx](http://blogs.msdn.com/dsimmons/archive/2008/02/17/context-lifetimes-dispose-or-reuse.aspx)), Simmons recommended:

- Add large numbers of new entities to the persistence store in batches; don't attempt to load, for example, 200,000 entities in a single transaction.
- Create a new `ObjectContext` on each Web service request or Web page post to keep the service or project stateless.
- Minimize load on the database by Windows form applications by caching slowly changing lookup tables locally, keeping the `ObjectContext` open for the entire session, and performing a refresh operation on lookup data periodically to prevent stale data from associations.
- For combinations of transient and static data, maintain the static data in an open `ObjectContext` and periodically refresh it, but detach the transient data.

You can't serialize the `ObjectContext` to persist it to an ASP.NET page's `ControlState` or application's `SessionState` property, but you can persist `EntityObject` instances without associated data to local

## Part V: Implementing the ADO.NET Entity Framework

caches. The .NET 3.5 SP1 version of the `DataContractSerializer` lets you persist object graphs that include cyclic relationships to XML streams for Web services or local XML files for persistence.

## Comparing the Performance of LINQ to Entities and Out-of-Band SQL Updates

The following table compares the average time in seconds to complete with the `NwindUpdatesCS.sln` and `NwindUpdatesVB.sln` projects the CUD operations described in the “Operation” column:

Operation	Out-of-Band SQL	LINQ to Entities Batched	LINQ to Entities Individual
Update 2155 Order_Detail items	0.043	3.683	N/A
Add 576 Customer items	0.593	0.556	1.382
Delete 576 Customer items	0.142	0.148	N/A

Figure 13-1 is the Windows form for the `NwindUpdatesCS.sln` project version.

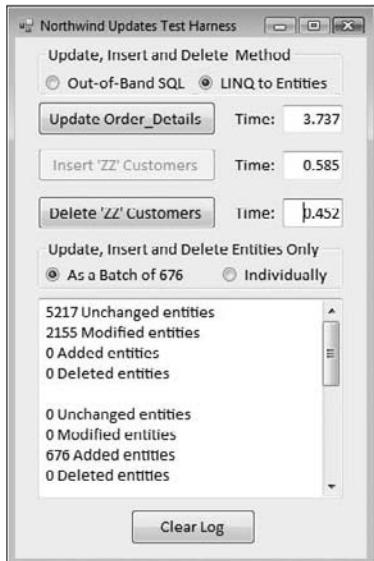


Figure 13-1

## Chapter 13: Updating Entities and Complex Types

---

Following are additional details of the test process:

- ❑ The project was built and run with F5 before collecting each data point to minimize the effects of establishing a connection to the local SQL Server 2005 Express instance and local data caching on the results. Values are the average of five tests.
- ❑ The code updates Order Detail rows and `Order_Detail` entities by adding 1 to the `Quantity` column/property. A second iteration, which is not included in the elapsed time values, reverts the update.
- ❑ There were no significant differences in elapsed times between the C# and VB project versions.
- ❑ The fairer comparison is the LINQ to Entities Individual test, which invokes the `SaveChanges(false)` method 576 times, rather than the LINQ to Entities Batches time, which invokes `SaveChanges(false)` once. The same would be true for the other two tests, but the exception noted previously prevents collecting the data. Thus, the similarity in elapsed time for adding and deleting the Customer entities is misleading.

The extreme (100:1) difference in execution time of the updates to 2,155 Order Details records and `Order_Details` entities is surprising and deserves further investigation. SQL Server Profiler shows that the 2,155 update queries, such as the following, execute in 1.604 seconds:

### T-SQL

```
exec sp_executesql N'update [dbo].[Order Details]
set [Quantity] = @0
where (([OrderID] = @1) and ([ProductID] = @2))
',N'@0 smallint,@1 int,@2 int',@0=15,@1=10250,@2=65
```

Iterating the `foreach` loop consumes about 0.118 second. Therefore, it appears that SQL Server takes about 1.871 seconds to commit the transaction. This appears excessive to me, when the following out-of-band update runs in 0.043 second, even considering that T-SQL is set-based and LINQ is instance-based.

### T-SQL

```
UPDATE [Order Details] SET Quantity = Quantity +1;
```

## Managing Optimistic Concurrency Conflicts

Databases that enable multiple users to update the same tables must decide on a strategy to resolve conflicts when two or more users update the same row with different column values. One option, called *pessimistic concurrency*, requires an updating user to lock the row to update, read its current column values, complete the edit, and unlock the row. One of the primary problems with pessimistic concurrency is that the lock prevents other users from reading the row, which causes transactions that involve the row to stall or roll back.

The alternative, *optimistic concurrency*, doesn't require a read operation and lock prior to attempting an update or deletion. In this case, it's assumed that the editing user has a local copy of the data to be edited — for EF, this an `EntityObject` instance that contains *original values*. The `UPDATE Table SET Column = Value` or `DELETE FROM Table` SQL statement includes a `WHERE` clause with a primary key constraint to specify the row to be updated and original value constraints for columns that are significant

## Part V: Implementing the ADO.NET Entity Framework

to concurrency management. If another user has edited any of these columns after the editing user obtained the original values, the database returns 0 as the number of records updated or deleted. Depending on the client's data provider, you can test for an unexpected 0 value or the provider throws an exception — an `OptimisticConcurrencyException` for ADO.NET data providers.

*An alternative to the use of original property values is to add a column with the timestamp data type to the table and entity, then perform a similar comparison between the value saved by the entity and that in the table currently. The timestamp field leads to simpler WHERE clause constraints but adds a persistence artifact to the entity, which violates the principle of “persistence ignorance” and would exclude the use of database engines that don't have a timestamp or similar data type. The “Updating an Entity Instance and Managing Concurrency with a Timestamp Property” section describes use of timestamps by stored procedures.*

To complete the update or deletion after detection of a concurrency conflict, it's necessary to invoke the `DataContext.Refresh(RefreshMode.ClientWins, EntityObject)` method to overwrite the local original values with values from the database and call the `DataContext.SaveChanges()` method again.

## Enabling Optimistic Concurrency Management for Entity Properties

EF's optimistic concurrency management feature is disabled by default; the `EntityName.PropertyNameConcurrencyMode` value for all entity properties defaults to `None`. Enabling participation in concurrency conflict tests requires setting the property's `ConcurrencyMode` value to `Fixed` in the EDM Designer. Figure 13-2 shows the Properties sheet for the `NorthwindModel.Customer` EntityType with the `ConcurrencyMode` property value set to `Fixed`.

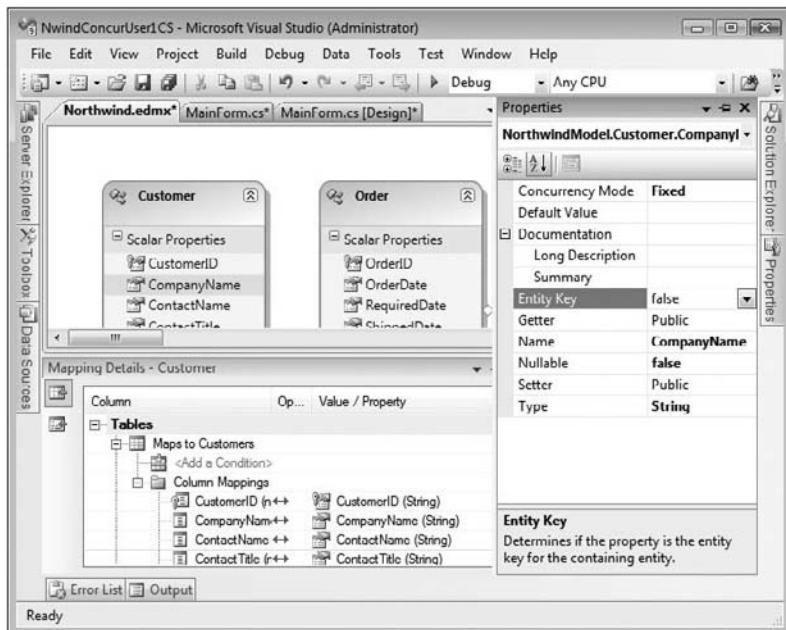


Figure 13-2

Each property whose `ConcurrencyMode` value you set to `Fixed` adds a criterion to the WHERE clause. For example, if you've set the `Customer` entity's `CompanyName`, `ContactName`, and `ContactTitle` to participate in concurrency tests, and change the value of `CompanyName` from `Bogus Software, Inc.` to `Bogus Software Corp.`, the update SQL statement will appear similar to this example captured with SQL Profiler:

### T-SQL

```
exec sp_executesql N'update [dbo].[Customers]
set [CompanyName] = @0
where (((([CustomerID] = @1) and ([CompanyName] = @2)) and
      ([ContactName] = @3)) and ([ContactTitle] = @4))
',N'@0 nvarchar(40),@1 nchar(5),@2 nvarchar(40),@3 nvarchar(30),
@4 nvarchar(30)',@0=N'Bogus Software Corp.',@1=N'BOGUS',
@2=N'Bogus Software, Inc.',@3=N'Joe Bogus',@4=N'President'
```

## ***Implementing Optimistic Concurrency Management with Code***

The `NwindConcurrencyCS.sln` and `NwindConcurrencyVB.sln` sample projects in the `\WROX\ADONET\Chapter13\CS` and `..\VB` folders demonstrate code that implements optimistic concurrency management for updates.

### C# 3.0

```
private void btnSaveChanges_Click(object sender, EventArgs e)
{
    Customer bogus = (from b in ctxNwind.Customers
                       where b.CustomerID == "BOGUS"
                       select b).FirstOrDefault();
    if (bogus != null)
    {
        try
        {
            if (isNameChanged)
            {
                bogus.CompanyName = txtCompanyName.Text;
                txtLog.Text += "CompanyName changed to '" +
                    bogus.CompanyName + "'\r\n";
            }
            if (isContactChanged)
            {
                bogus.ContactName = txtContactName.Text;
                txtLog.Text += "ContactName changed to '" +
                    bogus.ContactName + "'\r\n";
            }
            if (isTitleChanged)
            {
                bogus.ContactTitle = txtContactTitle.Text;
                txtLog.Text += "ContactTitle changed to '" +
                    bogus.ContactTitle + "'\r\n";
            }
        }
    }
}
```

## Part V: Implementing the ADO.NET Entity Framework

---

```
        }
        int changes = ctxNwind.SaveChanges(true);
    }
catch (OptimisticConcurrencyException ocEx)
{
    // Concurrency conflict occurred
    string exc = ocEx.Message;
    string msg = "A concurrency conflict occurred.\r\n\r\n" +
        "Click Yes to overwrite the other user's changes.\r\n\r\n" +
        "Click No to overwrite your changes with the other user's " +
        "changes.\r\n\r\n" +
        "Click Cancel to abandon your attempted edits.";
    DialogResult result =
        MessageBox.Show(msg, "Northwind Concurrency Test Harness",
            MessageBoxButtons.YesNoCancel, MessageBoxIcon.Question);
    if (result == DialogResult.Yes)
    {
        // Keep client updated values
        ctxNwind.Refresh(RefreshMode.ClientWins, bogus);
        ctxNwind.SaveChanges(true);
        txtLog.Text += "Your changes were persisted.";
    }
    else if (result == DialogResult.No)
    {
        // Reject client updated values
        ctxNwind.Refresh(RefreshMode.StoreWins, bogus);
        txtLog.Text += "Your changes were overwritten.";
        LoadBogusTextBoxes(bogus);
    }
}
btnVerify.Enabled = true;
}
}
```

### VB 9.0

```
Private Sub btnSaveChanges_Click(ByVal sender As Object,
    ByVal e As EventArgs) Handles btnSaveChanges.Click
    Dim bogus As Customer = (From b In ctxNwind.Customers _
        Where b.CustomerID = "BOGUS" _
        Select b).FirstOrDefault()
    If bogus IsNot Nothing Then
        Try
            If isNameChanged Then
                bogus.CompanyName = txtCompanyName.Text
                txtLog.Text &= "CompanyName changed to '" & _
                    bogus.CompanyName & "'" & vbCrLf
            End If
        Catch ex As Exception
            txtLog.Text &= "Exception: " & ex.Message
        End Try
    End If
End Sub
```

## Chapter 13: Updating Entities and Complex Types

---

```
End If
If isContactChanged Then
    bogus.ContactName = txtContactName.Text
    txtLog.Text &= "ContactName changed to '" & _
        bogus.ContactName & "'" & vbCrLf
End If
If isTitleChanged Then
    bogus.ContactTitle = txtContactTitle.Text
    txtLog.Text &= "ContactTitle changed to '" & _
        bogus.ContactTitle & "'" & vbCrLf
End If
Dim changes As Integer = ctxNwind.SaveChanges(True)
Catch ocEx As OptimisticConcurrencyException
    ' Concurrency conflict occurred
    Dim exc As String = ocEx.Message
    Dim msg As String = "A concurrency conflict occurred." _
        & vbCrLf & vbCrLf & _
        "Click Yes to overwrite the other user's changes." _
        & vbCrLf & vbCrLf & _
        "Click No to overwrite your changes with the other user's changes." _
        & vbCrLf & vbCrLf & _
        "Click Cancel to abandon your attempted edits."
    Dim result As DialogResult = MessageBox.Show(msg,
        "Northwind Concurrency Test Harness", MessageBoxButtons.YesNoCancel,
        MessageBoxIcon.Question)
    If result = DialogResult.Yes Then
        ctxNwind.Refresh(RefreshMode.ClientWins, bogus)
        ctxNwind.SaveChanges(True)
        txtLog.Text &= "Your changes were persisted."
    ElseIf result = DialogResult.No Then
        ctxNwind.Refresh(RefreshMode.StoreWins, bogus)
        txtLog.Text &= "Your changes were overwritten."
        LoadBogusTextBoxes(bogus)
    End If
End Try
btnVerify.Enabled = True
End If
End Sub
```

The catch block captures `OptimisticConcurrencyException`s and the `MessageBox` enables the user to select between overwriting the other user's entries by refreshing the entity with the `ClientWins` option (Yes), or accepting the other user's entries with the `StoreWins` option (No). Figure 13-3 shows the message that occurs when a simulated other user (left instance) changes the `CompanyName` value to `Bogus Software Corp.` and you (right instance) attempt to change `ContactTitle` to `President`.

## Part V: Implementing the ADO.NET Entity Framework

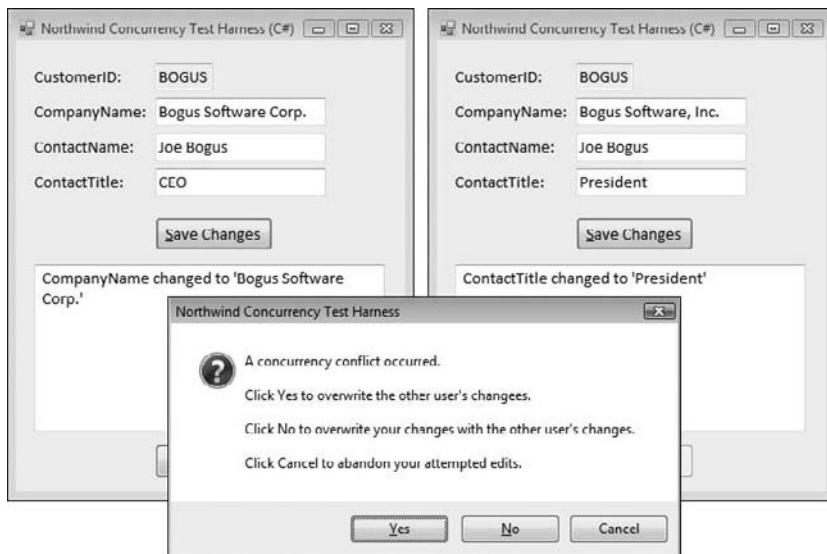


Figure 13-3

Figure 13-4 shown the result of you clicking Yes to overwrite the other user's (original) contact title.

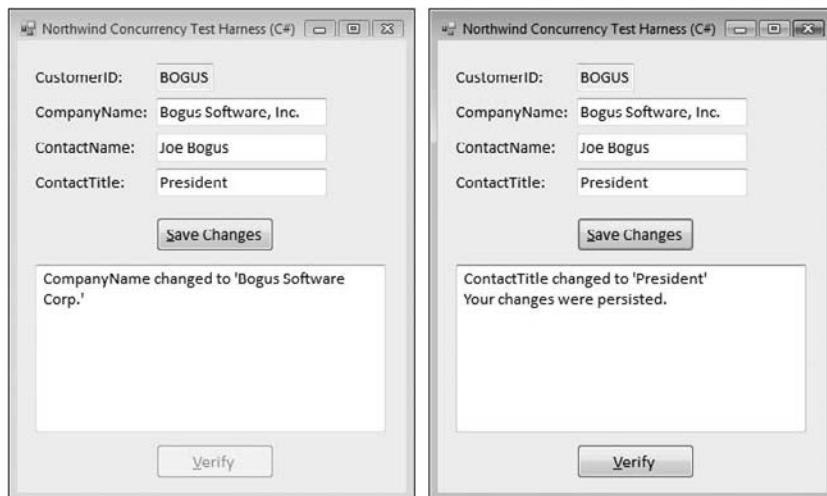


Figure 13-4

# Performing CRUD Operations with Stored Procedures

It's a common IT department policy to limit database access by users and most applications to stored procedures for all CRUD operations. The primary benefit of substituting stored procedures for SQL batch operations is increased security. SQL Server requires individual users and applications or groups to be assigned explicit EXECUTE permission on each stored procedure; other commercial and open-source RDBMSs have similar restrictions. The major drawback of stored procedures is loss of eSQL query features. However, read-only views can provide a reasonable compromise between security and flexibility for many types of enterprise data.

*SQL Server stored procedures no longer offer significant performance benefits because, effective with version 7.0, the database engine no longer compiles queries into stored procedures. Instead, parameterized queries are cached and reused by multiple-client applications. The parameterized SQL queries that EF generates from eSQL and executes with sp\_executesql are generally accepted to be immune to common SQL-injection attacks.*

The “Function Subelements and Subgroups” section and later sections of Chapter 9, “Defining Storage, Conceptual, and Mapping Layers,” describe the Function and FunctionImport groups of the \*.edmx file’s StorageModels group. Fortunately, the Entity Data Model Wizard supports tables, views, stored procedures and user-defined functions as data sources.

*The sample EF applications for the following sections are C:\WROX\ADONET\Chapter13\CS\NwindProdsSProcsCS.sln and ... \VB\NwindProdsSProcsVB.sln. The samples use the Northwind Categories, Products and Suppliers tables only. The script to create the required stored procedures for the applications is C:\WROX\ADONET\Chapter13\NwindProductsStoredProcs.sql and should be run from SQL Server Management Studio [Express] against your default instance of the Northwind sample database before running the sample EF applications.*

The following sections describe how to use the Entity Data Model Wizard to add the three tables, one view, and seventeen stored procedures to the model’s design surface, use the Create Function Import feature to substitute three stored procedures for eSQL queries to hydrate the Categories, Products, and Suppliers EntitySets, and use the Map Entity to Functions feature to specify the appropriate user stored procedure to create, update, or delete members of the three EntitySets.

## Creating the Initial Data Model

A simple EF project consisting of Category, Product, and Supplier entities created from the Northwind Categories, Products, and Suppliers tables is sufficient to demonstrate substituting stored procedures for eSQL or LINQ to Entities queries and autogenerated T-SQL INSERT, UPDATE and DELETE queries. With the stored procedures set up as described in the preceding note, start a new C# or VB Windows form project, add a new ADO.NET Entity Data Model template named Northwind.edmx, and click Add to start the Entity Data Model Wizard.

Accept the Generate from Database option, click Next, select the Northwind.dbo or equivalent connection, accept the default NorthwindEntities as the connection string name, and click Next to populate the Choose Your Database Objects’ Tables, Views, and Stored Procedures nodes. Mark the

## Part V: Implementing the ADO.NET Entity Framework

Tables node's Categories, Products, and Suppliers check boxes, optionally mark the Views node's uvw\_GetAllProducts check box, and mark all 22 of the Stored Procedures node's check boxes for names that have a `usp_*` prefix (see Figure 13-5).

*The uvw\_GetAllProducts view adds an EntitySet with a DefiningQuery that isn't used. The view is to demonstrate how the EDM runtime infers a primary key by concatenating the required properties (ProductID, ProductName and Discontinued).*

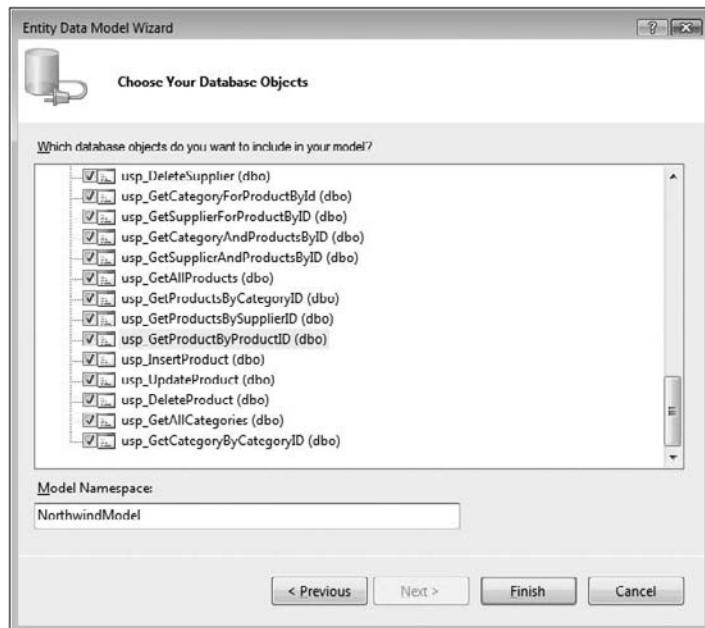


Figure 13-5

Accept NorthwindEntities as the namespace name, and click Finish to create the EDM with the Products entity and its associated Categories and Suppliers entities. For consistency with this book's entity naming conventions, singularize the entity names and the Product entity's Navigation Properties names. Finally, press F5 to build and run the project.

### **Adding FunctionImports to Populate the EntitySets**

FunctionImport elements in the ConceptualModels and Mappings group of the \*.edmx file connect to the StorageModels group's Function element that represents the `SELECT *` stored procedure. For example, the CSDL Function element for the `usp_GetAllProducts` stored procedure is:

#### **XML: CSDL**

```
<Function Name="usp_GetAllProducts" Aggregate="false" BuiltIn="false"
NiladicFunction="false" IsComposable="false"
ParameterTypeSemantics="AllowImplicitConversion" Schema="dbo" />
```

## Chapter 13: Updating Entities and Complex Types

You create the required CSDL and MSL entries by expanding the Model Browser's Stored Procedures node, right-clicking the stored procedure that you want to assign to a particular EntitySet, Products for this example, and selecting Create Function Import to open the New Function Import dialog (see Figure 13-6, which is a double-exposure).

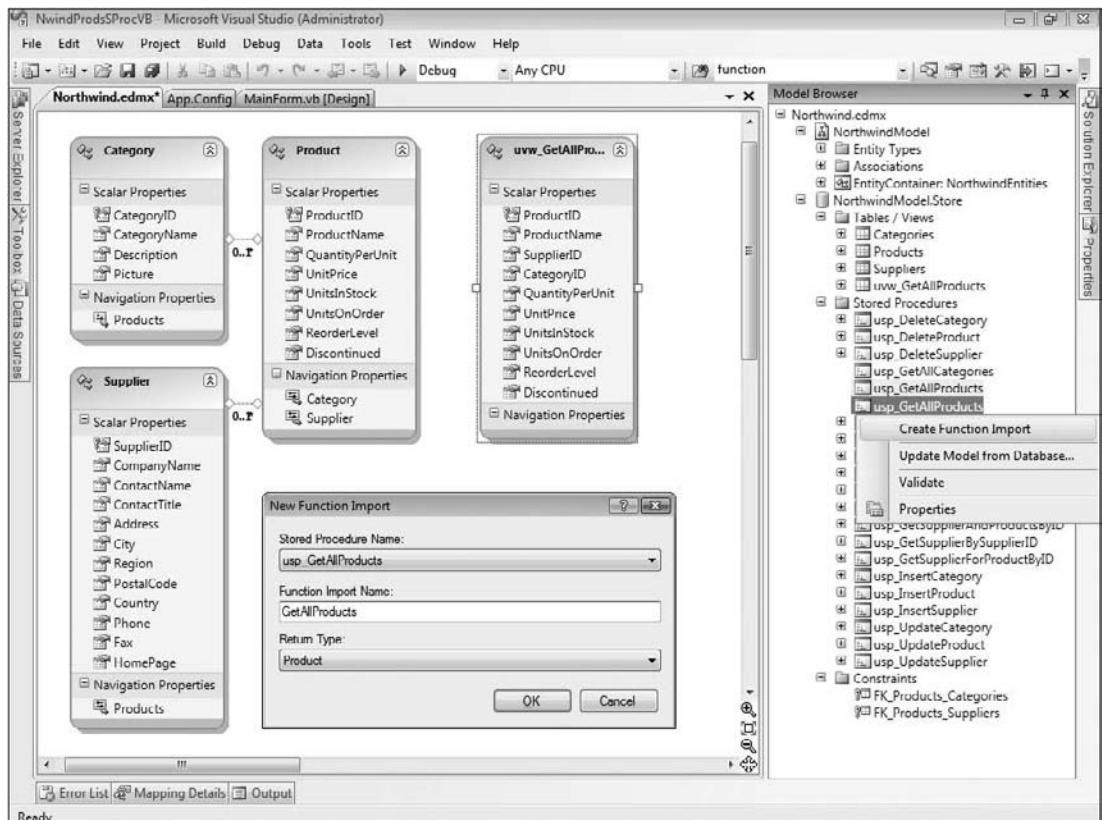


Figure 13-6

Change the Function Import Name property if you want and select the corresponding EntityObject type from the Return Type list, GetAllProducts and Product for this example. Clicking OK adds the following <FunctionImport> element to the CSDL section:

### XML: CSDL

```
<FunctionImport Name="GetAllProducts" EntitySet="Products"
    ReturnType="Collection(Self.Product)" />
```

and a second <FunctionImport> element to the MSL section:

### XML: MSL

```
<FunctionImportMapping FunctionImportName="GetAllProducts"
    FunctionName="NorthwindModel.Store.usp_GetAllProducts" />
```

## Part V: Implementing the ADO.NET Entity Framework

Repeat the preceding process for the `usp_GetAllCategories` and `usp_GetAllSuppliers` stored procedures.

*The sample projects also use the `usp_GetCategoryForProductID` and stored `usp_GetSupplierForProductID` stored procedure as `GetCategoryForProductID` and `GetSupplierForProductID` FunctionImports to supply associated many:one entities.*

## Assigning Insert, Update and Delete Stored Procedures to Entities

The Mapping Details pane with the Map Entity to Functions feature that you enable by clicking the  button (shown active in Figure 13-7) enables replacing parameterized SQL queries with `INSERT`, `UPDATE`, and `DELETE` stored procedures. If you specify a stored procedure for one of the three operations, you must assign the other two operations also; you can't mix and match an entity's CUD methods.

### Using a Stored Procedure to Insert a New Entity Instance

Inserting a new instance with a store procedure requires clicking the `<Select Insert Function>` item and selecting the appropriate `INSERT` stored procedure, `usp_InsertProduct`, from a drop-down list, which fills the Property list with matching parameter names. Navigation properties (foreign-key names) don't appear in the Property list, so you must click the corresponding row, open another drop-down list, and select the appropriate `TableName.Columnname` value for the corresponding foreign key, `Suppliers .SupplierID` for this example (see Figure 13-7).

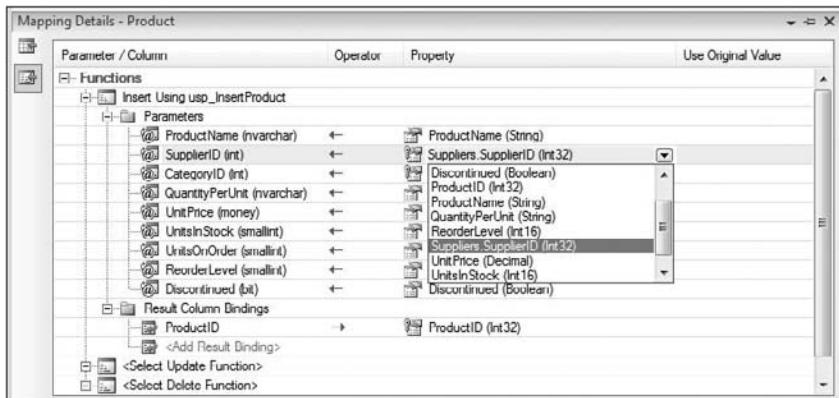


Figure 13-7

Repeat the preceding process for other empty navigation property/foreign key entries, `Category .CategoryID` in this instance.

The Result Column Bindings item, the `ProductID` property for this example (refer to Figure 13-7), holds the primary-key value assigned by the `int identity` (autoincrementing) data type that's used for all Northwind primary keys except the `Customers` table's `CustomerID` field. The following sample stored procedure includes highlighted code to return the `ProductID` value.

## T-SQL

```
CREATE PROCEDURE usp_InsertProduct (
    @ProductName nvarchar(40), @SupplierID int = 1, @CategoryID int = 1,
    @QuantityPerUnit nvarchar(20) = 'Insert package data',
    @UnitPrice money = 10.00, @UnitsInStock smallint = 0,
    @UnitsOnOrder smallint = 0, @ReorderLevel smallint = 0, @Discontinued bit = 0)
AS
SET NOCOUNT ON

INSERT INTO Products (ProductName, SupplierID, CategoryID, QuantityPerUnit,
    UnitPrice, UnitsInStock, UnitsOnOrder, ReorderLevel, Discontinued)
VALUES (@ProductName, @SupplierID, @CategoryID, @QuantityPerUnit, @UnitPrice,
    @UnitsInStock, @UnitsOnOrder, @ReorderLevel, @Discontinued);

SELECT SCOPE_IDENTITY() AS ProductID WHERE @@ROWCOUNT > 0;
SET NOCOUNT OFF;
GO
```

## **Updating an Entity Instance and Managing Concurrency with Original Values**

The conventional approach for updating entities and managing concurrency conflict resolution use stored procedures with original-value parameters added for all properties that are significant for comparison to persisted values. Components that autogenerated update and deletion code, such as the ASP.NET SqlDataSource control, use this approach because it doesn't require a change to the table's schema to implement concurrency management features. For example, the following usp\_UpdateProduct\_CM stored procedure provides original values for all fields except ProductID:

## T-SQL

```
CREATE PROCEDURE usp_UpdateProduct_CM (
    @ProductID int, @ProductName nvarchar(40), @SupplierID int,
    @CategoryID int, @QuantityPerUnit nvarchar(20), @UnitPrice money,
    @UnitsInStock smallint = 0, @UnitsOnOrder smallint = 0,
    @ReorderLevel smallint = 0, @Discontinued bit = 0,
    @OriginalProductName nvarchar(40), @OriginalSupplierID int,
    @OriginalCategoryID int, @OriginalQuantityPerUnit nvarchar(20),
    @OriginalUnitPrice money, @OriginalUnitsInStock smallint = 0,
    @OriginalUnitsOnOrder smallint = 0, @OriginalReorderLevel smallint = 0,
    @OriginalDiscontinued bit = 0
)
AS
SET NOCOUNT ON

UPDATE Products SET
    ProductName = @ProductName, SupplierID = @SupplierID, CategoryID = @CategoryID,
    QuantityPerUnit = @QuantityPerUnit, UnitPrice = @UnitPrice,
    UnitsInStock = @UnitsInStock, UnitsOnOrder = @UnitsOnOrder,
    ReorderLevel = @ReorderLevel, Discontinued = @Discontinued

WHERE ProductID = @ProductID AND @OriginalProductName = ProductName
    AND @OriginalSupplierID = SupplierID AND @OriginalCategoryID = CategoryID
    AND @OriginalQuantityPerUnit = QuantityPerUnit
```

## Part V: Implementing the ADO.NET Entity Framework

```
AND @OriginalUnitPrice = UnitPrice AND @OriginalUnitsInStock = UnitsInStock  
AND @OriginalUnitsOnOrder = UnitsOnOrder  
AND @OriginalReorderLevel = ReorderLevel  
AND @OriginalDiscontinued = Discontinued  
  
SET NOCOUNT OFF;  
GO
```

The Mapping Details pane for the corresponding update `FunctionImport` has a line for each parameter, including the `@OriginalFieldName` parameters. Supplying the original values to these parameters requires matching ...`FieldName` with the appropriate property name in the line's drop-down list and marking the Use Original Value check box, as shown in Figure 13-8.

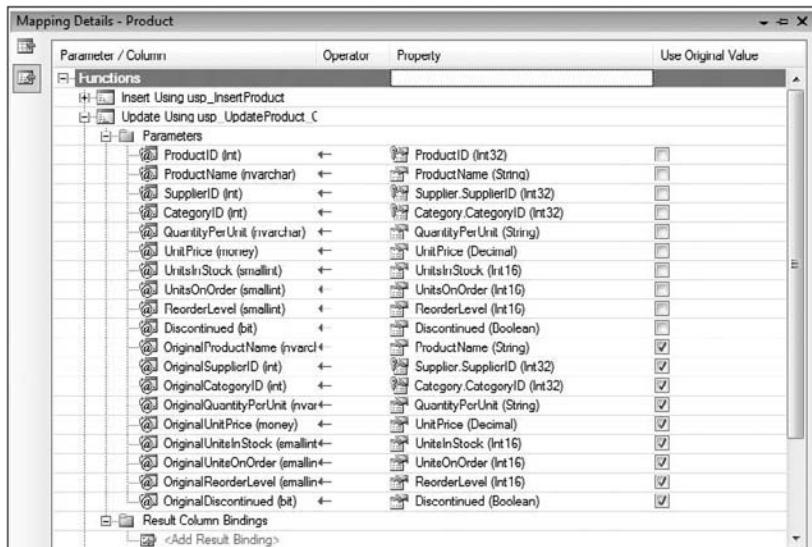


Figure 13-8

Following is the prebuilt parameterized query generated by the `NwindProdsSProcVB.sln` project (with original values emphasized) when you click the sample project's Edit Project button after the Load Projects and Add Project button:

### T-SQL

```
exec [dbo].[usp_UpdateProduct_CM] @ProductID=207,  
@ProductName=N'Edited Product',@SupplierID=29,@CategoryID=2,  
@QuantityPerUnit=N'36 250-ml bottles',@UnitPrice=20.0000,@UnitsInStock=4,  
@UnitsOnOrder=15,@ReorderLevel=10,@Discontinued=0,  
@OriginalProductName=N'New Product',@OriginalSupplierID=1,@OriginalCategoryID=1,  
@OriginalQuantityPerUnit=N'24 250-ml bottles',@OriginalUnitPrice=15.0000,  
@OriginalUnitsInStock=5,@OriginalUnitsOnOrder=15,@OriginalReorderLevel=10,  
@OriginalDiscontinued=0
```

Handling `OptimisticConcurrencyExceptions` with stored procedures uses the same approach as that described in the earlier “Implementing Optimistic Concurrency Management with Code” section.

*The NwindProdsSProcVB.sln sample project uses the `usp_UpdateProduct_CM` stored procedure to provide original values.*

## Updating an Entity Instance and Managing Concurrency with a Timestamp Property

If you’re starting a new (“greenfield”) project or have the authority to change table’s schema to add a field of the `timestamp` or equivalent data type, you can save considerable design effort and gain a slight performance improvement by substituting a single `OriginalTimestamp` value for the multiple `OriginalFieldName` values (see Figure 13-9) and setting the `Timestamp` property’s `Concurrency Mode` to `Fixed`.

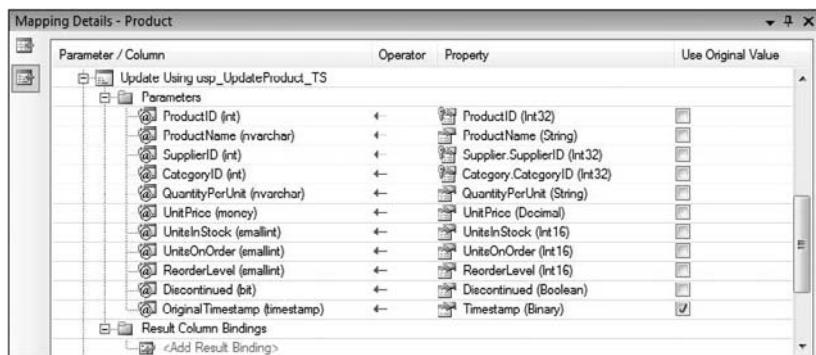


Figure 13-9

Here’s the much simpler code to create the `usp_UpdateProduct_TS` stored procedure with a timestamp field added to the `Products` table:

### T-SQL

```
CREATE PROCEDURE usp_UpdateProduct_TS (
    @ProductID int, @ProductName nvarchar(40), @SupplierID int, @CategoryID int,
    @QuantityPerUnit nvarchar(20), @UnitPrice money, @UnitsInStock smallint = 0,
    @UnitsOnOrder smallint = 0, @ReorderLevel smallint = 0, @Discontinued bit = 0,
    @OriginalTimestamp timestamp)

AS
SET NOCOUNT ON

UPDATE Products SET
    ProductName = @ProductName, SupplierID = @SupplierID, CategoryID = @CategoryID,
    QuantityPerUnit = @QuantityPerUnit, UnitPrice = @UnitPrice,
    UnitsInStock = @UnitsInStock, UnitsOnOrder = @UnitsOnOrder,
    ReorderLevel = @ReorderLevel, Discontinued = @Discontinued
WHERE ProductID = @ProductID AND [Timestamp] = @OriginalTimestamp
SET NOCOUNT OFF;
GO
```

## Part V: Implementing the ADO.NET Entity Framework

---

The NwindProdsSProcsC#.sln sample project uses the usp\_UpdateProduct\_TS stored procedure to support a timestamp column. Code in the sample project adds and drops the Timestamp column during the Form\_Load and Form\_Closing events.

### **Deleting Entity Instances with Stored Procedures**

Stored procedures for deleting entity instances must include the same parameters for associated entities as the procedures for inserting or updating them. Thus the usp\_DeleteProduct stored procedure requires the following statement:

#### **T-SQL**

```
CREATE PROCEDURE usp_DeleteProduct (
    @ProductID int,
    @SupplierID int,
    @CategoryID int)
AS
SET NOCOUNT ON

DELETE FROM Products
WHERE ProductID = @ProductID
AND SupplierID = @SupplierID
AND CategoryID = @CategoryID;

SET NOCOUNT OFF;
```

Deleting a previously added `Product` instance generates this statement:

```
exec [dbo].[usp_DeleteProduct] @ProductID=271, @SupplierID=1, @CategoryID=1
```

WHERE criteria for original-value and timestamp-based optimist concurrency management are similar to those for UPDATE operations.

## **Working with Complex Types**

Complex types implement entity properties that support multiple properties but don't have an entity key value. Martin Fowler, the author of *Patterns of Enterprise Application Architecture*, and others, call complex types *value objects* because they are based on sets of values rather than an entity-key reference. Complex types have no identity outside of the entity that contains them; they are considered identical to other complex types if all their property values are equal.

The most common examples of complex types are addresses and currency. For example, a `BillingAddress` type might have `StreetAddress1`, `StreetAddress2`, `PostalAddress`, `City`, `Region`, `PostalCode`, and `Country` properties, where `StreetAddress1` and `StreetAddress2` or `PostalAddress` are nullable, but other properties are required. Similarly, a `ShippingAddress` might have the same fields but only allow `StreetAddress2` and `PostalAddress` to have null values, if shipment isn't made by a postal service. A `Currency` type usually has `CurrencyCode` and `Units` properties, neither of which is nullable.

*Unfortunately, EF v1's complex types don't support nullable properties, so you must substitute empty strings to prevent exceptions on attempting to open an ObjectContext instance.*

## Modeling a Complex Type

Entity Framework v1's EDM Designer doesn't support defining complex types graphically, so you must use the XML Editor to roll your own entries for the CSDL and MSL sections of the \*.edmx file. The \*.edmx file's SSDL section doesn't require modification. The first step, however, is to use the Entity Data Model Wizard to autogenerate the \*.edmx file from the requisite database tables. For this example, the Customer entity's Address, City, Region, PostalCode and Country properties are moved to same-named properties of a BillingAddress complex type.

*Once you make the required changes to the CSDL and MSL sections, you won't be able to open the EDM Designer window. So it's a good practice to add all the entities your project needs to the designer before modeling your first complex type.*

### Modifying the CSDL Section

You must add a ComplexProperty element to the EntityType group. This element specifies the complex type's assigned name and type, such as shown here for a BillingAddress type:

#### XML: CSDL

```
<Property Name="BillingAddress" Type="Self.ComplexAddress" Nullable="false" />
```

The Self prefix and Nullable="false" attribute/value pair are required. EF v1 doesn't support nullable complex types.

You then add a ComplexType subgroup to the Schema group and move the appropriate Property elements to the ComplexProperty group, as shown here for the ComplexAddress type:

#### XML: CSDL

```
<ComplexType Name="ComplexAddress">
    <Property Name="Address" Type="String" MaxLength="60" Nullable="false" />
    <Property Name="City" Type="String" MaxLength="15" Nullable="false" />
    <Property Name="Region" Type="String" MaxLength="15" />
    <Property Name="PostalCode" Type="String" MaxLength="10" />
    <Property Name="Country" Type="String" MaxLength="15" Nullable="false" />
</ComplexType>
```

A single instance of the preceding ComplexAddress type can be referenced by other entities that have similar complex type requirement, such as Supplier, Order and Employee.

## Part V: Implementing the ADO.NET Entity Framework

---

### Modifying the MSL Section

The C-S mapping section requires wrapping the `<ScalarProperty>` elements for the complex type's properties with `<ComplexProperty Name = "ComplexPropertyName">` tags, as shown by the highlighted lines in the following example:

#### XML: MSL

```
<EntitySetMapping Name="Customers">
  <EntityTypeMapping TypeName="IsTypeOf(NorthwindModel.Customer)">
    <MappingFragment StoreEntitySet="Customers">
      <ScalarProperty Name="CustomerID" ColumnName="CustomerID" />
      <ScalarProperty Name="CompanyName" ColumnName="CompanyName" />
      <ScalarProperty Name="ContactName" ColumnName="ContactName" />
      <ScalarProperty Name="ContactTitle" ColumnName="ContactTitle" />

      <ComplexProperty Name ="BillingAddress">
        <ScalarProperty Name="Address" ColumnName="Address" />
        <ScalarProperty Name="City" ColumnName="City" />
        <ScalarProperty Name="Region" ColumnName="Region" />
        <ScalarProperty Name="PostalCode" ColumnName="PostalCode" />
        <ScalarProperty Name="Country" ColumnName="Country" />
      </ComplexProperty>
      <ScalarProperty Name="Phone" ColumnName="Phone" />
      <ScalarProperty Name="Fax" ColumnName="Fax" />
    </MappingFragment>
  </EntityTypeMapping>
</EntitySetMapping>
```

After you make the preceding modifications and build your project you might see two error messages: "Complex types are not supported in the designer" and "Property BillingAddress is not mapped or used in a condition." These errors are spurious and don't cause operational problems.

### Reusing a ComplexType Definition

In addition to aliasing complex types, you also can alias property names. The mapping for the Orders EntitySet that aliases the ShipName property to the more common ShipTo name and substitutes the ShipAddress . . . ShipCountry columns for the ShippingAddress ComplexAddress type is highlighted in the following group:

#### XML: MSL

```
<EntitySetMapping Name="Orders">
  <EntityTypeMapping TypeName="IsTypeOf(NorthwindModel.Order)">
    <MappingFragment StoreEntitySet="Orders">
      <ScalarProperty Name="OrderID" ColumnName="OrderID" />
      <ScalarProperty Name="EmployeeID" ColumnName="EmployeeID" />
      <ScalarProperty Name="OrderDate" ColumnName="OrderDate" />
      <ScalarProperty Name="RequiredDate" ColumnName="RequiredDate" />
      <ScalarProperty Name="ShippedDate" ColumnName="ShippedDate" />
      <ScalarProperty Name="ShipVia" ColumnName="ShipVia" />
      <ScalarProperty Name="Freight" ColumnName="Freight" />
```

```
<ScalarProperty Name="ShipTo" ColumnName="ShipName" />
<ComplexProperty Name ="ShippingAddress">
    <ScalarProperty Name="Address" ColumnName="ShipAddress" />
    <ScalarProperty Name="City" ColumnName="ShipCity" />
    <ScalarProperty Name="Region" ColumnName="ShipRegion" />
    <ScalarProperty Name="PostalCode" ColumnName="ShipPostalCode" />
    <ScalarProperty Name="Country" ColumnName="ShipCountry" />
</ComplexProperty>
</MappingFragment>
</EntityTypeMapping>
</EntitySetMapping>
```

The modified CSDL group for the preceding mapping is:

### XML: CSDL

```
<EntityType Name="Order">
    <Key>
        <PropertyRef Name="OrderID" />
    </Key>
    <Property Name="OrderID" Type="Int32" Nullable="false" />
    <Property Name="EmployeeID" Type="Int32" />
    <Property Name="OrderDate" Type="DateTime" />
    <Property Name="RequiredDate" Type="DateTime" />
    <Property Name="ShippedDate" Type="DateTime" />
    <Property Name="ShipVia" Type="Int32" />
    <Property Name="Freight" Type="Decimal" Precision="19" Scale="4" />
    <Property Name="ShipTo" Type="String" MaxLength="40" Unicode="true"
        FixedLength="false" />
    <Property Name="ShippingAddress" Type="Self.ComplexAddress" Nullable = "false" />
    <NavigationProperty Name="Customers"
        Relationship="NorthwindModel.FK_Orders_Customers"
        FromRole="Orders" ToRole="Customers" />
</EntityType>
```

Access to properties of complex types for data retrieval or update uses conventional dot notation:  
EntityName.ComplexTypeName.PropertyName.

The \WROX\ADONET\Chapter13\NwindComplexTypesCS\ComplexTypesCS.sln and ...\\NwindComplexTypesVB\ComplexTypesVB.sln sample projects implement the BillingAddress ComplexAddress for Customer entities and ShippingAddress ComplexAddress for Order entities. The projects perform an out-of-band replacement of null address values by empty strings and add a BOGUS customer on startup (Form\_Load) and reverse the process on shutdown (Form\_Closing). Bound text boxes let you edit the BOGUS customer, save changes, and refresh the data to verify the change. Selecting customers other than BOGUS disable editing and display the customer's Order EntitySets based on a match of ShipTo with CompanyName property values. The two sample projects use simple Object Data Source databinding, which is one of the topics of the next chapter.

# Summary

It's a relatively simple task to build an O/RM tool for retrieving raw data from a disk-based store and creating object instances with the data. That's why there are so many choices of O/RM tools for .NET, Java, Ruby, Python, and other popular programming platforms and languages. However, you don't need an O/RM tool to hydrate objects; code to set property values from relational resultsets is trivial. Tracking object additions, updates, and deletions, as well as persisting these changes to the data store is a considerably more complex process. The difficulty of managing this process is undoubtedly one of the primary reasons that EF v1 doesn't implement an eSQL DML dialect.

Commercial and open-source O/RMs differ greatly in their approaches to the design and implementation of object caching, identification, change tracking, and change persistence, as well as optimistic concurrency management conflict resolution and support for CUD stored procedures. Some O/RMs, such as LLBLGen Pro, store the change-tracking data in the entity instance. EF's `ObjectContext`, like LINQ to SQL's `DataContext`, provides a single, top-level object to handle all CRUD tasks. The `ObjectContext.ObjectStateManager` holds the state of entity instances. EF's approach minimizes — but doesn't eliminate — addition of object-persistence artifacts to domain models.

This chapter began with a brief discussion of the `ObjectContext.ObjectStateManager` and its `EntityKey` property for instance identification and  `EntityState` enumeration for determining if an `EntityObject` instance is `Unchanged`, `Modified`, `Added` or `Deleted`. Code samples covered updating, deleting, and adding instances, as well as persisting changes to a relational data store. EF is database-agnostic, so there are no modifications required to the code shown if you substitute another EF-enhanced managed ADO.NET data provider for the default `SqlClient`. Other update-related topics discussed data freshness and validation, and optimizing the `ObjectContext`'s lifetime. A comparison of execution times using LINQ to Entities and out-of-band SQL DML commands demonstrated that the latter approach offers much better performance for bulk updates. Examples of EF's optimistic concurrency conflict management compared the original property values with the current values of database fields.

Many organizations require the use of stored procedures for all database access, so sections covered using `FunctionImports` elements to populate `EntitySets`, assigning stored procedures to CUD operations, and concurrency management with stored procedures. Conflict management with stored procedures compared the original property value approach with the simpler row timestamp comparison method.

The chapter closed with a lengthy discussion of complex types, which are named hierarchical groups of properties that cannot exist independently of instances. The examples used the canonical complex type, `Address`, to substitute for the billing address fields of the `Customer` entity and the `Order` entity's shipping address fields.

# 14

## Binding Entities to Data-Aware Controls

Data-binding the Entity Framework's generic `ObjectContext.EntitySet` collections and their members to data-aware Windows form and ASP.NET server controls follows Microsoft's original Wizard-based drag-and-drop pattern for typed DataSets. In most cases, you use similar databinding methodology with typed DataSets, LINQ to SQL and Entity Framework (EF) data sources. Windows forms require that you add a node for the top-level (master) `EntitySet` to the Data Sources window; if the relationships/associations have been specified correctly, the master `EntitySet` will have nested `EntityCollection(s)` for one:many associations. It's a common practice to drag the master node to the form to create a details view with a `BindingNavigator`, `BindingSource` and multiple bound `TextBox` controls. Nested `EntityCollection` nodes commonly generate `DataGridView` controls to enable selection and editing of associated entities. ASP.NET's `EntityDataSource` control lets developers bind data-aware, sorted, paged `GridView` or `FormView` controls to an `EntitySet` by specifying a few control properties.

A "demoware" stigma afflicts Windows and, to a lesser degree, Web form projects that rely on autogenerated databound controls. The implication is that "real developers don't use wizards and drag-and-drop techniques" to create industrial-quality applications. However, automating the databinding process can markedly increase programmer productivity, especially for small to moderate-size projects. ADO.NET's Data Sources window makes creating a simple, hierarchical database administration or data entry program with a top-level `EntitySet` as its data source a quick and easy task. Creating data entry and editing pages with ASP.NET involves a bit more work, because you need to specify an `EntityDataSource` control for each entity. However, you can autogenerate a multi-page administrative site for an entire database with ASP.NET Dynamic Data in a few minutes. (ASP.NET Dynamic Data is one of the subjects of Chapter 15.)

This chapter's Windows form projects use `BindingSource` components as well as bound `DataGridView` and `ComboBox` controls; Web form projects use prebuilt ASP.NET `EntityDataSource`, `GridView`, and `FormView` server controls.

*This chapter's sample projects are in the \WROX\ADONET\Chapter14\C# and ... \VB folders. The sample projects are intended to demonstrate EF programming techniques but aren't representative of production-grade applications. For example, most projects don't include exception handling to make it easier to determine the line of source code that throws the exception. In addition, examples run UI and execute databinding operations on the same thread; a better approach is to execute databinding operations with a BackgroundWorker component to prevent a Windows form's UI from freezing in the event of a delay or failure to create a connection. Another option is to reduce the ConnectTimeout property value to less than 15 seconds when connecting to an intranet database.*

# Binding Windows Form Controls to Entities with Object Data Sources

If you've created the canonical master/detail Windows form with a typed DataSet as the data source, you know the drill for doing the same with an Entity Data Model as the data source:

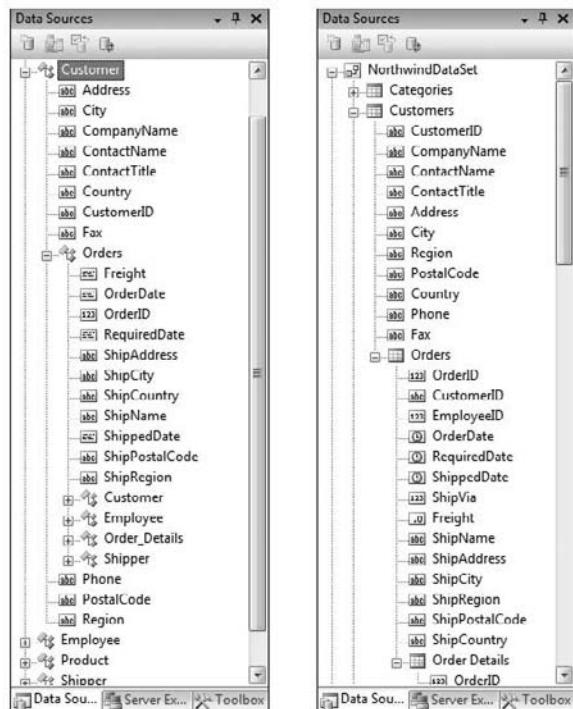
1. Add a new ADO.NET Entity Data Model, and select the tables to add with the EDM Wizard.
2. Singularize EDM entity and many:one association names.
3. Add a new Data Source to open the Data Source Configuration Wizard, but select the Object data source type instead of the Database data source type in the first wizard dialog.
4. Expand the nodes in the assembly list and select the top-level entity node, Customers for the Northwind sample database, to populate the master, details, and subdetails controls in the second wizard dialog instead of selecting the database connection and tables in the second and third wizard dialogs.
5. Add nodes for the remaining entities to the Data Sources window and expand them (see Figure 14-1).



Figure 14-1

Chapter 14: Binding Entities to Data-Aware Controls

Figure 14-2 compares the visible individual data sources for the NorthwindEntities EDM in the left pane and the NorthwindDataSet in the right pane. NorthwindEntities contains EntitySets (Customer, Employee and Product), EntityCollections (Customer.Orders and Order.Order\_Details), and EntityReferences (Orders.Customer, Orders.Employee, Orders.Shipper). The Northwind DataSet shows DataTables (Category and Customer are visible) and one:many relationships (Customer.Orders and Orders.Order\_Details are visible) but doesn't treat relationships as bidirectional (reciprocal). Therefore, no nodes that correspond to reciprocal many:one relationships appear in the DataSet pane. On the other hand, foreign-key values — such as CustomerID, EmployeeID, and ShipVia — are missing from NorthwindEntities.



**Figure 14-2**

*LINQ to SQL exposes nodes for Entities, EntityRefs (one:many) and EntitySets (many:one), as well as foreign-key values. Foreign-key values simplify code for updating associations when adding or replacing dependent entities. An EDM Association is immutable. You can add or remove Association elements from an AssociationSet, but you can't modify them.*

## **Using the Load(), Include(), and Attach() Methods for Shaping Object Graphs**

Populating bound controls with the property values of dependent entities associated with the project's master or target entity is one of the primary tasks in developing master/detail forms. As mentioned in the "Enabling Deferred or Eager Loading of Associated Entities" section of Chapter 12, EF doesn't support implicit lazy loading of an `EntitySet`'s associated objects. Therefore, it's the developers'

## Part V: Implementing the ADO.NET Entity Framework

responsibility to add code to populate the missing related entities from the persistence store. A simple misstep in this process can result in downloading to the client most or all of the contents of a giant, multi-table database.

For example, creating the Northwind sample database's `Customers` `EntitySet` with its one:many `EntityCollections`, as well their `EntityCollections` and many:one `EntityReferences`, would result in instantiating more than 3,200 objects. A real-world version of a large Web retailer's online transaction processing (OLTP) database for order processing might attempt to generate millions of objects under the same conditions. If the OLTP database for a manufacturer includes recent customer purchases, a query for a single customer's orders might return thousands of heavyweight objects.

There are only a few cases in which detail or subdetail entities don't require filtering. An example is an order's line items; the user doesn't gain a complete view of an order without being able to inspect *all* line items. On the other hand, users ordinarily need to view a customers last few or unshipped orders only. Thus, associated orders must be filtered, ordered, or both to accommodate the application's users. This pattern is common to a wide range of other entities, such as medical records, airline passenger reservations, and the like.

Figure 14-3 shows the C# version of this section's `WinFormEdmDataBindingCS.sln` and `WinFormEdmDataBindingVB.sln` sample project displaying the `Customer` entity selected by `CustomerID` value in the combo box, the Customer's last five orders, and the `Order_Detail` set for the last Order. Clicking the Cancel button refreshes the data for the selected Customer without saving changes.

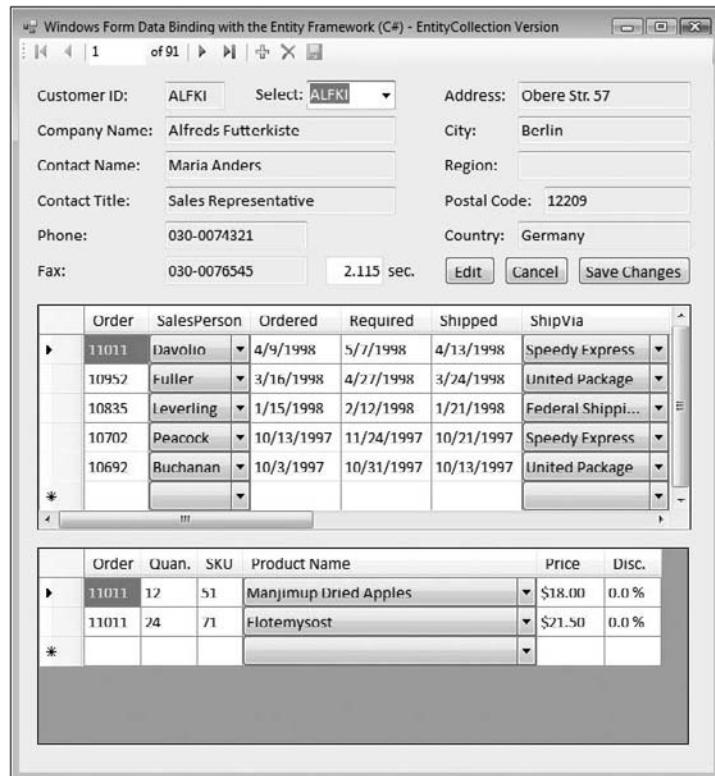


Figure 14-3

The unbound SalesPerson and ShipVia combo boxes in added columns change the EmployeeID:Order and ShipperID:Order association.

*The “Enabling Deferred or Eager Loading of Associated Entities” topic of Chapter 12 describes methods for deferred and eager loading of EntityCollections and EntityReferences in the context of ObjectQueries.*

### **Deferred Loading of Associated Objects with the Load() Method**

The following examples invoke the `Load([MergeOption.ChosenOption])` method on the `Customer.Orders` EntityCollection. A LINQ to Entities query loads the `Order_Details` instances for the last five `Order` instances, as in these examples with the sort and filter expressions highlighted:

#### **C# 3.0**

```
NorthwindEntities ctxNwind = new NorthwindEntities()
List<Customer> custList = new List<Customer>();
// Return and iterate the main EntitySet
var custs = from c in ctxNwind.CustomerSet
            where c.Country == "USA"
            select c;
foreach (Customer c in custs)
{
    if (!c.Orders.IsLoaded)
    {
        // Load Order_Details for the last five members
        // of the EntityCollection if not cached
        c.Orders.Load(MergeOption.AppendOnly);
        var orders = (from o in c.Orders
                      orderby o.OrderDate descending
                      select o).Take(5);
        foreach (Order o in orders)
        {
            // Defer-load each EntityRef and EntityCollection
            o.CustomerReference.Load(MergeOption.AppendOnly);
            o.EmployeeReference.Load(MergeOption.AppendOnly);
            o.ShipperReference.Load(MergeOption.AppendOnly);
            o.Order_Details.Load(MergeOption.AppendOnly);
        }
        custList.Add(c);
    }
}
```

*It's not necessary to apply the `IsLoaded` test to the `Order` entities' associated `Order_Details` objects because the `Order` is loaded once only.*

## Part V: Implementing the ADO.NET Entity Framework

### VB 9.0

```
Dim ctxNW As New NorthwindEntities
Dim custList As New List(Of Customer)()
' Return and iterate the main EntitySet
Dim custs = From c In ctxNW.CustomerSet _
    Where c.Country = "USA" _
    Select c
For Each c As Customer In custs
    If (Not c.Orders.IsLoaded) Then
        ' Load Order_Details for the last five members
        ' of the EntityCollection if not cached
        c.Orders.Load(MergeOption.AppendOnly);
        Dim orders = (From o In c.Orders _
            Order By o.OrderDate Descending _
            Select o).Take(5)
        For Each o As Order In orders
            ' Defer-load each EntityReference and EntityCollection
            o.CustomerReference.Load(MergeOption.AppendOnly)
            o.EmployeeReference.Load(MergeOption.AppendOnly)
            o.ShipperReference.Load(MergeOption.AppendOnly)
            o.Order_Details.Load(MergeOption.AppendOnly)
        Next o
    End If
    custList.Add(c);
Next c
```

The problem with this approach is that a roundtrip to the database server is required for each Customer instance, Order set, EntityRef, and EntityCollection. Another issue is that all Orders for each Customer are loaded, but Order\_Details members and, potentially, their Product EntityReferences, are loaded only for the last five Orders.

*The “Updating Associated EntitySets or EntityObjects” topic of Chapter 13 describes the optional MergeOption enumeration. MergeOption.AppendOnly is the default; you can’t use MergeOption.NoTracking to load associated objects.*

### Eager Loading of Multiple Associated Objects with the `Include()` Method

The ObjectContext doesn't support a direct equivalent to LINQ to SQL's `DataContext.LoadOptions` collection and `DataLoadOption` object. LINQ to SQL lets you combine the `DataLoadOption`'s `LoadWith()` and `AssociateWith()` methods to eager-load filtered EntitySets with a single SQL query. You can reduce the number of queries against the database server to one at the expense of losing the capability to filter and order associated entities during loading by invoking the chained `Include("IncludePath")` methods, as in the following examples:

### C# 3.0

```
NorthwindEntities ctxNW = new NorthwindEntities();
List<Customer> custList = null;
// Fill the List<Customer> with all Orders and Details
custList = (from c in ctxNW.CustomerSet
            .Include("Orders.Order_Details")
```

```
where c.Country == "USA"
select c).ToList();
```

### VB 9.0

```
Dim ctxNW As New NorthwindEntities()
Dim custList As List(Of Customer) = Nothing
' Fill the List<Customer> with all Orders and Details
custList = (From c In ctxNW.CustomerSet. _
Include("Orders.Order_Details") _
Where c.Country = "USA" _
Select c).ToList()
```

The `Include()` method associates each member of the path with the higher-level entity. This approach is satisfactory for shallow object graphs with a small number of object instances at each hierarchical level. It's also applicable to applications where the savings in connection overhead justify issuing a single request that retrieves unused instances.

### Deferred Loading of Associated Objects with the Attach() Method

The EF's `EntityCollection.Attach()` method is required to prevent adding unneeded objects to the `ObjectContext`. This method returns an `ObjectQuery<T>` that contains the same entities the corresponding `Load()` method invocation would return and populates the two `AssociationEnds`. The `OrderByDescending()` and `Take(5)` methods (highlighted in the code examples) are chained to the `ObjectQuery<T>` to return the last five orders in descending order:

### C# 3.0

```
NorthwindEntities ctxNW = new NorthwindEntities();
List<Customer> custList = New List<Customer>();
// Fill the List<Customer> with all Orders and Details
var custs = from c in ctxNW.CustomerSet
            where c.Country == "USA"
            select c;

foreach (Customer c in custs)
{
    if (c.Orders.Count == 0)
    {
        c.Orders.Attach(c.Orders.CreateSourceQuery() .
OrderByDescending(o => o.OrderDate).Take(5));
        foreach (Order o in c.Orders)
            o.Order_Details.Attach(o.Order_Details.CreateSourceQuery() .
Where(d => d.OrderID == o.OrderID));
        custList.Add(c);
    }
}
```

## Part V: Implementing the ADO.NET Entity Framework

---

### VB 9.0

```
Dim ctxNW As New NorthwindEntities()
Dim custList As List(Of Customer) = Nothing
' Fill the List<Customer> with all Orders and Details
Dim custs = From c In ctxNW.CustomerSet _
    Where c.Country = "USA" _
    Select c
For Each c As Customer In custs
    If c.Orders.Count = 0 Then
        c.Orders.Attach(c.Orders.CreateSourceQuery() . _
            OrderByDescending(Function(o) o.OrderDate).Take(5))
        For Each o As Order In c.Orders
            o.Order_Details.Attach(o.Order_Details.CreateSourceQuery() . _
                Where(Function(d) d.OrderID = o.OrderID))
        Next o
        custList.Add(c)
    End If
Next c
```

You'll probably find that the `EntityCollection.Attach()` method is the most efficient means to populate object graphs for bound controls, especially when you select an individual top-level entity instance to edit, as illustrated by the next section, or add new top-level and detail instances predominately.

## Selecting the Active Top-Level and Associated Entity Instances

The sample applications eager-load and fill `List<Customer>`, `List<Employee>`, `List<Shipper>`, and `List<Product>` lists during the `Form_Load` event because these generic lists contain relatively few entity instances compared with `OrdersSet` and `Order_DetailsSet`. `List<Customer>` serves as the data source for the `customerBindingSource` and `cboCustomer` combo box that serves as the `Customer` entity picker.

*If Customer was a heavyweight entity, you could minimize resource consumption by populating the ComboBox by a projection of CustomerID, or CustomerID and CompanyName if your ComboBox displayed the latter. However, selecting the Customer must execute a query to return the entire entity to assign as the CustomerBindingSource.DataSource property value.*

Here's the sample projects' code to set the selected `Customer.Item` as the `customerBindingSource.Current` value:

### C# 3.0

```
// Load a customer, orders, and order_details from a pick list
private void cboCustomer_SelectedIndexChanged(object sender, EventArgs e)
{
    if (isLoaded) // true after loading cboCustomer.List
    {
        timer.Reset();
        timer.Start();
        customerBindingSource.MoveFirst();
```

```
    isLoaded = false;
    for (int i = 0; i < customerBindingSource.Count; i++)
    {
        Customer cust = (Customer)customerBindingSource.Current;
        Customer sel = (Customer)cboCustomer.SelectedValue;
        if (cust.CustomerID == sel.CustomerID)
        {
            isLoaded = true;
            customerBindingSource_CurrentChanged(null, null);
            // Display Order and details load time
            timer.Stop();
            txtTime.Text =
                (timer.ElapsedMilliseconds / 1000D).ToString("0.000");
            break;
        }
        else
            customerBindingSource.MoveNext();
    }
}
```

### VB 9.0

```
' Load a customer, orders, and order_details from a pick list
Private Sub cboCustomer_SelectedIndexChanged(ByVal sender As Object, _
                                         ByVal e As EventArgs)
    If isLoaded Then ' True after loading cboCustomer.List
        timer.Reset()
        timer.Start()
        CustomerBindingSource.MoveFirst()
        isLoaded = False
    For i As Integer = 0 To CustomerBindingSource.Count - 1
        Dim cust As Customer = CType(CustomerBindingSource.Current, Customer)
        Dim sel As Customer = CType(cboCustomer.SelectedValue, Customer)
        If cust.CustomerID = sel.CustomerID Then
            isLoaded = True
            CustomerBindingSource_CurrentChanged(Nothing, Nothing)
            ' Display Order and details load time
            timer.Stop()
            txtTime.Text =
                (timer.ElapsedMilliseconds / 1000R).ToString("0.000")
            Exit For
        Else
            CustomerBindingSource.MoveNext()
        End If
    Next i
End If
End Sub
```

*The customerBindingSource.Current value and cboCustomers.Value are loosely coupled to accommodate insertions or deletions of Customer entities after initially loading the ComboBox list as well as moving the Current value with the navigation buttons.*

## Part V: Implementing the ADO.NET Entity Framework

---

The customerBindingSource\_CurrentChanged() event handler adds associated Orders and Order\_Details entity instances to the orderBindingSource.Items and order\_DetailBindingSource.Items collection with the code shown here:

### C# 3.0

```
// Load Orders/Order_Details on demand
private void customerBindingSource_CurrentChanged(object sender, EventArgs e)
{
    if (isLoaded) // true after loading cboCustomer.List
    {
        // Get the current Customer instance
        Customer cust = (Customer)customerBindingSource.Current;

        // Attach the last five Orders and Order_Details if not cached
        if (cust.Orders.Count == 0)
        {
            cust.Orders.Attach(cust.Orders.CreateSourceQuery() .
                OrderByDescending(o => o.OrderDate).Take(5));
            foreach (Order o in cust.Orders)
                o.Order_Details.Attach(o.Order_Details.CreateSourceQuery() .
                    Where(d => d.OrderID == o.OrderID));
        }

        // Provide foreign-key values for Employee and ShipVia ComboBoxes
        ordersBindingSource.MoveFirst();
        int row = 0;
        foreach (Order ord in ordersBindingSource)
        {
            if (ord.Employee != null)
                orderDataGridView.Rows[row].Cells[1].Value =
                    ord.Employee.EmployeeID;
            if (ord.Shipper != null)
                orderDataGridView.Rows[row].Cells[5].Value =
                    ord.Shipper.ShipperID;
            row++;
        }
    }
}
```

### VB 9.0

```
' Load Orders/Order_Details on demand
Private Sub customerBindingSource_CurrentChanged(ByVal sender As Object, _
                                                 ByVal e As EventArgs)
    If isLoaded Then ' true after loading cboCustomer.List
        ' Get the current Customer instance
        Dim cust As Customer = CType(customerBindingSource.Current, Customer)

        ' Attach the last five Orders and Order_Details if not cached
        If cust.Orders.Count = 0 Then
            cust.Orders.Attach(cust.Orders.CreateSourceQuery() ._
                OrderByDescending(Function(o) o.OrderDate).Take(5))
            For Each o As Order In cust.Orders
                o.Order_Details.Attach(o.Order_Details.CreateSourceQuery() .
```

```
        Where(Function(d) d.OrderID = o.OrderID))  
    Next o  
End If  
  
' Provide foreign-key values for Employee and ShipVia ComboBoxes  
ordersBindingSource.MoveFirst()  
Dim row As Integer = 0  
For Each ord As Order In ordersBindingSource  
    If ord.Employee IsNot Nothing Then  
        orderDataGridView.Rows(row).Cells(1).Value = _  
            ord.Employee.EmployeeID  
    End If  
    If ord.Shipper IsNot Nothing Then  
        orderDataGridView.Rows(row).Cells(5).Value = _  
            ord.Shipper.ShipperID  
    End If  
    row += 1  
Next ord  
End If  
End Sub
```

*You must write code to obtain the foreign-key values of EDM entities. LINQ to SQL exposes foreign-key values as first class properties of dependent entities.*

## Using Unbound ComboBoxes to Specify Associations

The preceding code adds foreign-key values to synchronize the unbound EmployeeIDComboBox's and ShipperIDComboBox's Display property values with the current Order.Employee and Order.Shipper property values; a null value for added orders has an empty Display property value. Members of EntityCollections are immutable; to change the association, you must delete the original entity instance and add the new one. You can change the EntityReference.Value property to point to the new entity instance.

### **Adding Event Handlers for ComboBoxColumns**

You must wire up SelectedValueChanged event handlers for unbound ComboBox controls. Following is the code to delegate event handling for the orderDataGridView's two unbound combo boxes:

#### **C# 3.0**

```
// Intercept changes to Orders ComboBoxes  
private void orderDataGridView_EditingControlShowing(object sender,  
    DataGridViewEditingControlShowingEventArgs e)  
{  
    // Handle ComboBox changes only  
    if (e.Control is DataGridViewComboBoxEditingControl)  
    {  
        // Detect EmployeeIDComboBox in column 2  
        DataGridViewComboBoxEditingControl cbo =  
            e.Control as DataGridViewComboBoxEditingControl;  
        if (orderDataGridView.SelectedCells[0].ColumnIndex == 1)  
        {
```

## Part V: Implementing the ADO.NET Entity Framework

---

```
// Remove an existing event-handler, if present, to avoid
// adding multiple handlers when the editing control is reused
cbo.SelectedValueChanged -=
    new EventHandler(EmployeeIDComboBox_SelectedValueChanged);

// Add the event handler
cbo.SelectedValueChanged +=
    new EventHandler(EmployeeIDComboBox_SelectedValueChanged);
}

// Detect ShipperIDComboBox in column 6
else if (orderDataGridView.SelectedCells[0].ColumnIndex == 5)
{
    cbo.SelectedValueChanged -=
        new EventHandler(ShipperIDComboBox_SelectedValueChanged);

    cbo.SelectedValueChanged +=
        new EventHandler(ShipperIDComboBox_SelectedValueChanged);
}
}
```

### VB 9.0

```
' Intercept changes to Orders ComboBoxes
Private Sub orderDataGridView_EditingControlShowing(ByVal sender As Object,
    ByVal e As DataGridViewEditingControlShowingEventArgs)
    ' Handle ComboBox changes only
    If TypeOf e.Control Is DataGridViewComboBoxEditingControl Then
        ' Detect EmployeeIDComboBox in column 2
        Dim cbo As DataGridViewComboBoxEditingControl =
            TryCast(e.Control, DataGridViewComboBoxEditingControl)
        If orderDataGridView.SelectedCells(0).ColumnIndex = 1 Then
            ' Remove an existing event-handler, if present, to avoid
            ' adding multiple handlers when the editing control is reused
            RemoveHandler cbo.SelectedValueChanged, _
                AddressOf EmployeeIDComboBox_SelectedValueChanged

            ' Add the event handler
            AddHandler cbo.SelectedValueChanged, _
                AddressOf EmployeeIDComboBox_SelectedValueChanged

        ' Detect ShipperIDComboBox in column 6
        ElseIf orderDataGridView.SelectedCells(0).ColumnIndex = 5 Then
            RemoveHandler cbo.SelectedValueChanged, _
                AddressOf ShipperIDComboBox_SelectedValueChanged

            AddHandler cbo.SelectedValueChanged, _
                AddressOf ShipperIDComboBox_SelectedValueChanged
        End If
    End If
End Sub
```

### **Updating Association Sets with the SelectedIndexChanged Event Handler**

The following event-handling code adds or changes the `Employee.Orders` collection and the `Order.EmployeeReference` depending on settings of the `EmployeeID` combo box:

#### **C# 3.0**

```
// Update Order.Employee and Employee.Orders
private void EmployeeIDComboBox_SelectedIndexChanged(object sender, EventArgs e)
{
    if (((ComboBox)sender).SelectedValue is Int32)
    {
        // New EmployeeID value from ComboBox selection
        int newId = (int)((ComboBox)sender).SelectedValue;

        // Currently selected Order
        Order order = (Order)orderBindingSource.Current;
        if (order.Employee == null || newId != order.Employee.EmployeeID)
        {
            // Remove the old Order entity from the Employee.Orders collection
            if (order.Employee != null)
                order.Employee.Orders.Remove(order);

            // Get the new Employee entity
            Employee newEmpl = ctxNwind.EmployeeSet.
                Where(m => m.EmployeeID == newId).FirstOrDefault();

            // Add the new Order to the Employee.Orders collection
            newEmpl.Orders.Add(order);

            // Set the EntityReference to the new Employee
            order.EmployeeReference.Value = newEmpl;
        }
    }
}
```

#### **VB 9.0**

```
' Update Order.Employee and Employee.Orders
Private Sub EmployeeIDComboBox_SelectedIndexChanged(ByVal sender As Object, ByVal e As EventArgs)
    If TypeOf (CType(sender, ComboBox)).SelectedValue Is Int32 Then
        ' New EmployeeID value from ComboBox selection
        Dim newId As Integer = CInt(Fix(CType(sender, ComboBox).SelectedValue))

        ' Currently selected Order
        Dim order As Order = CType(orderBindingSource.Current, Order)
        If order.Employee Is Nothing OrElse newId <> order.Employee.EmployeeID Then
            ' Remove the old Order entity from the Employee.Orders collection
            If order.Employee IsNot Nothing Then
                order.Employee.Orders.Remove(order)
            End If
        End If
    End If
End Sub
```

## Part V: Implementing the ADO.NET Entity Framework

---

```
' Get the new Employee entity
Dim newEmpl As Employee = ctxNwind.EmployeeSet. _
    Where(Function(m) m.EmployeeID = newId).FirstOrDefault()

' Add the new Order to the Employee.Orders collection
newEmpl.Orders.Add(order)

' Set the EntityReference to the new Employee
order.EmployeeReference.Value = newEmpl
End If
End If
End Sub
```

Code for the ShipperID combo box is identical to the preceding except for object names.

*Exposing foreign-key values probably will be an option in EF v2.*

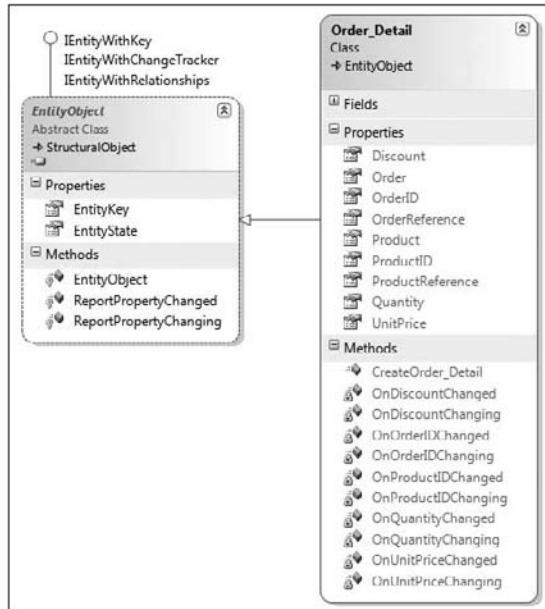
## **Setting Composite Primary-Key Members with Combo Boxes**

The EntityKey of objects generated from tables with composite primary keys incorporated values from all key columns. This means that *instances* of these objects are immutable and requires that you delete and re-create the instance and its AssociationSet(s) when changing the scalar value of a primary-key member. Composite primary keys usually are the result of using a combination of *natural keys* rather than a *surrogate key* to identify table rows. *Natural keys* combine property values of the entities represented by rows to generate a unique value. *Surrogate keys* have no direct association with the entity or its properties and are assigned by the database engine; for SQL Server, the most common types of surrogate keys are autoincrementing int identity and uniqueidentifier GUID columns. There is considerable controversy as to the relative desirability of these two types of primary keys, but they commonly are found in online transaction processing (OLTP) database schemas.

The Northwind sample database's Order Details table is one of the better examples of the use of a composite primary key. Combining OrderID and ProductID foreign-key values from the Orders and Order Details tables provides a unique value to self-identify instances and prevents duplicating entries for the same product in a single order. An alternative is a combination of OrderID and a sequential ItemID value, but line item sequence is an artificial property that must be maintained by code, such as a database trigger, stored procedure, or external business rule.

## **Detecting Attempts to Change Composite Primary-Key Values**

Editing Order\_Detail entities might involve substituting one stock-keeping unit (SKU) for another, which involves changing a ProductID property value. All EntityObject partial classes in the ModelName.Designer.cs file include a set of methods on each property's setter to notify the ObjectContext's change-tracker that the property value has changes pending and, if the new value is valid, has completed the change. Figure 14-4 is a class diagram of the Order\_Detail EntityObject and its abstract base class that shows the OnEntityObjectChanging and OnEntityObjectChanged methods for each property.



**Figure 14-4**

Following is the change notification code from the sample project's `Northwind.Designer.cs` and `.vb` files for the `Order_Details.ProductID` property:

### C# 3.0

```
[global::System.Data.Objects.DataClasses.
EdmScalarPropertyAttribute(EntityKeyProperty=true, IsNullable=false)]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public int ProductID
{
    get
    {
        return this._ProductID;
    }
    set
    {
        this.OnProductIDChanging(value);
        this.ReportPropertyChanging("ProductID");
        this._ProductID = global::System.Data.Objects.DataClasses.
            StructuralObject.SetValidValue(value);
        this.ReportPropertyChanged("ProductID");
        this.OnProductIDChanged();
    }
}
private int _ProductID;

partial void OnProductIDChanging(int value);
partial void OnProductIDChanged();
```

## Part V: Implementing the ADO.NET Entity Framework

---

### VB 9.0

```
<Global.System.Data.Objects.DataClasses.  
EdmScalarPropertyAttribute(EntityKeyProperty:=True, IsNullable:=False),  
Global.System.Runtime.Serialization.DataMemberAttribute()> _  
Public Property ProductID() As Integer  
    Get  
        Return Me._ProductID  
    End Get  
    Set(ByVal value As Integer)  
        Me.OnProductIDChanging(value)  
        Me.ReportPropertyChanging("ProductID")  
        Me._ProductID = Global.System.Data.Objects.DataClasses. __  
            StructuralObject.SetValidValue(value)  
        Me.ReportPropertyChanged("ProductID")  
        Me.OnProductIDChanged()  
    End Set  
End Property  
  
Private _ProductID As Integer  
  
Partial Private Sub OnProductIDChanging(ByVal value As Integer)  
End Sub  
  
Partial Private Sub OnProductIDChanged()  
End Sub
```

The `OnProductIDChanging()` partial method could be used for data validation if the action was cancelable, which it's not in EF v1. Instead, the `this.ReportPropertyChanging("ProductID")` or `Me.ReportPropertyChanging("ProductID")` instructions throw a not-supported exception. However, you can throw your own `NotSupportedException` with a custom message by adding partial method code like the following:

### C# 3.0

```
public partial class Order_Detail  
{  
    partial void OnProductIDChanging(int value)  
    {  
        this.ProductIDChangingException(value);  
    }  
  
    void ProductIDChangingException(int value)  
    {  
        if (MainForm.origProductID > 0 && value != MainForm.origProductID)  
        {  
            string msg = "Changing a Product Name is not allowed because it's " +  
                "part of the EntityKey (object identifier).\\r\\n\\r\\nPlese delete " +  
                "the line item you want to change and then add the new line item.";  
            throw new NotSupportedException(msg);  
        }  
    }  
}
```

## VB 9.0

```
Partial Public Class Order_Detail
    Private Sub OnProductIDChanging(ByVal value As Integer)
        Me.ProductIDChangingException(value)
    End Sub

    Private Sub ProductIDChangingException(ByVal value As Integer)
        If MainForm.origProductID > 0 AndAlso value <> MainForm.origProductID Then
            Dim msg As String = "Changing a Product Name is not allowed " & _
                " because it's part of the EntityKey (object identifier)." & _
                & vbCrLf vbCrLf & "Please delete the line item you want " & _
                "to change and then add the new line item."
            Throw New NotSupportedException(msg)
        End If
    End Sub
End Class
```

The `order_DetailDataGridView_EditingControlShowing` event handler, which is the topic of the next section, sets the `origProductID` value, because the event fires when the user starts editing the control's content.

## **Enforcing Object Deletion and Addition Operations for Updates**

Automatically deleting the existing and adding a new object when the user makes a selection from the `ProductIDComboBox` or a similar editing control affects usability negatively because it's not easy to implement multiple level undo or redo for such operations. Thus, the approach taken by the sample applications is to notify the user by a message that manual deletion and addition of the two items is required. That message reads as follows:

*Changing a Product Name is not allowed because it's part of the object identifier (from ECS.)  
Please delete the line item you want to change and then add the new line item.*

The first line of defense against attempts to change `ProductID` values with the combo box is the `order_DetailDataGridView_EditingControlShowing` event handler, which delegates event handling for the `ProductIDComboBox` if the user is editing a new item or issues a warning when editing an existing item. Following is the event-handling procedure's code:

## C# 3.0

```
// Obtain the original ProductID value, add if 0, warn otherwise
private void order_DetailDataGridView_EditingControlShowing(object sender,
    DataGridViewEditingControlShowingEventArgs e)
{
    // Handle ComboBox changes only
    if (e.Control is DataGridViewComboBoxEditingControl)
    {
        // Detect ProductIDComboBox in column 3
        DataGridViewComboBoxEditingControl cbo =
            e.Control as DataGridViewComboBoxEditingControl;
        // Remove an existing event-handler, if present, to avoid
```

## Part V: Implementing the ADO.NET Entity Framework

---

```
// adding multiple handlers when the editing control is reused
cbo.SelectedValueChanged -=
    new EventHandler(ProductIDComboBox_SelectedValueChanged);
origProductID =
    ((Order_Detail)order_DetailBindingSource.Current).ProductID;
if (origProductID == 0)
    cbo.SelectedValueChanged +=
        new EventHandler(ProductIDComboBox_SelectedValueChanged);
else
{
    string msg = "Changing a Product Name is not allowed because it's " +
        "part of the object identifier (from ECS).\r\n\r\nPlease delete " +
        "the line item you want to change and then add the new line item.";
    MessageBox.Show(msg, msgTitle);
}
}
}
```

### VB 9.0

```
' Obtain the original ProductID value, add if 0, warn otherwise
Private Sub order_DetailDataGridView_EditingControlShowing(ByVal sender As Object,
    ByVal e As DataGridViewEditingControlShowingEventArgs)
    ' Handle ComboBox changes only
    If TypeOf e.Control Is DataGridViewComboBoxEditingControl Then
        ' Detect ProductIDComboBox in column 3
        Dim cbo As DataGridViewComboBoxEditingControl = _
            TryCast(e.Control, DataGridViewComboBoxEditingControl)
        ' Remove an existing event-handler, if present, to avoid
        ' adding multiple handlers when the editing control is reused
        RemoveHandler cbo.SelectedValueChanged, _
            AddressOf ProductIDComboBox_SelectedValueChanged
        origProductID = CType(order_DetailBindingSource.Current,
            Order_Detail)).ProductID
        If origProductID = 0 Then
            AddHandler cbo.SelectedValueChanged, _
                AddressOf ProductIDComboBox_SelectedValueChanged
        Else
            Dim msg As String = "Changing a Product Name is not allowed " + _
                "because it's part of the object identifier (from ECS)." & _
                vbCrLf & vbCrLf & "Please delete the line item you want " + _
                "to change and then add the new line item."
            MessageBox.Show(msg, msgTitle)
        End If
    End If
End Sub
```

The (from ECS) phrase in the message temporarily identifies the procedure that displays it.

### Selecting and Adding a New EntityObject and AssociationSets

The ProductIDComboBox\_SelectedIndexChanged event handler conducts a second check for an attempt to edit the ProductID value. If the user has selected the ProductName for a new Order\_Detail, the order\_DetailBindingSource.Current property value is an empty Order\_Detail object, so the code updates ProductID, SKU, and UnitPrice values. Associated entities are immutable, so users must delete an existing Order\_Detail object before adding a replacement.

#### C# 3.0

```
private void ProductIDComboBox_SelectedIndexChanged(object sender, EventArgs e)
{
    if (isLoaded && ((ComboBox)sender).SelectedValue is Int32)
    {
        // New ProductID value from ComboBox selection
        newProductID = (int)((ComboBox)sender).SelectedValue;
        string newProductName = (string)((ComboBox)sender).Text;
        if ((newProductID != origProductID) && origProductID > 0)
        {
            ((ComboBox)sender).SelectedValue = origProductID;
            string msg = "Changing a Product Name is not allowed because it's " +
                "part of the object identifier (from SVC).\r\n\r\nPlease delete " +
                "the line item you want to change and then add the new line item.";
            MessageBox.Show(msg, msgTitle);
            return;
        }

        if (newProductID > 0)
        {
            // New Order_Detail added by user in UI
            Order_Detail newDetail =
                (Order_Detail)order_DetailBindingSource.Current;

            // Get UnitPrice and default Quantity = 1
            var newProduct = (from p in productBindingSource.DataSource
                             as IEnumerable<Product>
                             where p.ProductID == newProductID
                             select p).FirstOrDefault();
            newDetail.ProductID = newProduct.ProductID;
            newDetail.UnitPrice = (Decimal)newProduct.UnitPrice;
            newDetail.Quantity = 1;

            // EndEdit() is required to prevent exceptions on following statements
            order_DetailBindingSource.EndEdit();
            orderDataGridView.EndEdit();

            // Add the new Order_detail to the Order.Order_Details and
            //     Product.Order_Details associations
            ((Order)orderBindingSource.Current).Order_Details.Add(newDetail);
            newProduct.Order_Details.Add(newDetail);
        }
    }
}
```

## Part V: Implementing the ADO.NET Entity Framework

---

```
// Add the Order and Product to the Order_Details.Order and
//      Order_Details.Product associations
newDetail.OrderReference.Value = (Order)orderBindingSource.Current;
newDetail.ProductReference.Value = newProduct;

// Reset the ProductIDs
origProductID = 0;
newProductID = 0;
}
}
}
```

### VB 9.0

```
Private Sub ProductIDComboBox_SelectedIndexChanged(ByVal sender As Object,
    ByVal e As EventArgs)
    If isLoaded AndAlso _
        TypeOf (CType(sender, ComboBox)).SelectedValue Is Int32 Then
        ' New ProductID value from ComboBox selection
        newProductID = CInt(Fix((CType(sender, ComboBox)).SelectedValue))
        Dim newProductName As String = CStr((CType(sender, ComboBox)).Text)
        If (newProductID <> origProductID) AndAlso origProductID > 0 Then
            CType(sender, ComboBox).SelectedValue = origProductID
            Dim msg As String = "Changing a Product Name is not allowed " & _
                "because it's part of the object identifier (from SVC)." & _
                vbCrLf & vbCrLf & "Please delete the line item you want " & _
                "to change and then add the new line item."
            MessageBox.Show(msg, msgTitle)
            Return
        End If

        If newProductID > 0 Then
            ' New Order_Detail added by user in UI
            Dim newDetail As Order_Detail = _
                CType(order_DetailBindingSource.Current, Order_Detail)

            ' Get UnitPrice default Quantity = 1
            Dim newProduct = (From p In TryCast(productBindingSource.DataSource, _
                IEnumerable(Of Product)) _
                Where) p.ProductID = newProductID _
                Select p).FirstOrDefault()
            newDetail.ProductID = newProduct.ProductID
            newDetail.UnitPrice = CType(newProduct.UnitPrice, Decimal)
            newDetail.Quantity = 1

            ' EndEdit() is required to prevent exceptions on following statements
            order_DetailBindingSource.EndEdit()
            orderDataGridView.EndEdit()

            ' Add the new Order_detail to the Order.Order_Details and
            '      Product.Order_Details associations
            CType(orderBindingSource.Current, Order).Order_Details.Add(newDetail)
            newProduct.Order_Details.Add(newDetail)
```

```
' Add the Order and Product to the Order_Details.Order and
' Order_Details.Product associations
newDetail.OrderReference.Value = _
    CType(orderBindingSource.Current, Order)
newDetail.ProductReference.Value = newProduct

' Reset the ProductIDs
origProductID = 0
newProductID = 0
End If
End If
End Sub
```

### Persisting Changes to the Data Store

The final step in the editing process is to persist the changes to the database and cause the ObjectStateManager to accept the changes by executing the following code in the `btnSaveChanges_Click` event handler:

#### C# 3.0

```
// Persist changes and accept changes after save
private void btnSaveChanges_Click(object sender, EventArgs e)
{
    try
    {
        // End all edits and invoke the SaveChanges method
        orderDataGridView.EndEdit();
        order_DetailDataGridView.EndEdit();
        customerBindingSource.EndEdit();
        orderBindingSource.EndEdit();
        order_DetailBindingSource.EndEdit();
        ctxNwind.SaveChanges(true);
        // Refresh the data (optional to verify and test)
        customerBindingSource_CurrentChanged(null, null);
    }
    catch (Exception ex)
    {
        string msg = ex.Message;
        if (ex.InnerException != null)
            msg += "\r\n\r\n" + ex.InnerException;
        if (msg.Contains("at System."))
        {
            msg = msg.Substring(0, msg.IndexOf("at System.") - 1);
        }
        MessageBox.Show(msg, msgTitle, MessageBoxButtons.OK,
            MessageBoxIcon.Exclamation);
    }
}
```

## Part V: Implementing the ADO.NET Entity Framework

---

### VB 9.0

```
' Persist changes and accept changes after save
Private Sub btnSaveChanges_Click(ByVal sender As Object, ByVal e As EventArgs)
    Try
        ' End all edits and invoke the SaveChanges method
        orderDataGridView.EndEdit()
        order_DetailDataGridView.EndEdit()
        customerBindingSource.EndEdit()
        orderBindingSource.EndEdit()
        order_DetailBindingSource.EndEdit()
        ctxNwind.SaveChanges(True)

        ' Refresh the data (optional to verify and test)
        customerBindingSource_CurrentChanged(Nothing, Nothing)
    Catch ex As Exception
        Dim msg As String = ex.Message
        If ex.InnerException IsNot Nothing Then
            msg &= Constants.vbCrLf & Constants.vbCrLf & ex.InnerException
        End If
        If msg.Contains("at System.") Then
            msg = msg.Substring(0, msg.IndexOf("at System.") - 1)
        End If
        MessageBox.Show(msg, msgTitle, MessageBoxButtons.OK, _
            MessageBoxIcon.Exclamation)
    End Try
End Sub
```

## Using the EntityDataSource with ASP.NET Server Controls

The EntityDataSource EntityDataSource server control derives from the `System.Web.UI.DataSourceControl` class and is EF's counterpart to LINQ to SQL's misnamed `LinqDataSource` control. EntityDataSource offers the following features to Web developers who need to display and edit property values of domain objects:

- ❑ Declarative two-way ASP.NET databinding to EDM objects
- ❑ Server-side sorting and paging without added code
- ❑ Caching of original entity property values in `ViewState` for concurrency conflict management of entity updates and deletions
- ❑ Optional caching of entity state in `ViewState`
- ❑ Flattening of complex types and `EntityReference` keys
- ❑ Support for ASP.NET Dynamic Data projects

*LinqDataSource is a misnomer because LINQ is the query technology underlying LINQ to SQL as well as other object types. Many developers who are new to C# 3.0 or VB 9.0 consider LINQ to be a synonym for its LINQ to SQL implementation and are unaware of LINQ's many other varieties: LINQ to Objects, LINQ to XML, LINQ to Entities, and dozens of third-party LINQ implementations.*

By default, the EntityDataSource's entities can't bind ASP.NET data-aware controls if they contain EntityReferences to associated entities, because most databound controls expect scalar property values. When you specify an `ObjectContext.EntitySet<T>` as the data source, a wrapper object implements the `ICustomTypeDescriptor` interface, which returns the foreign-key value that corresponds to the primary key of the associated entity. LINQ to SQL entities expose scalar foreign-key values and `EntityRefs`. The EntityDataSource's wrapper emulates the LinqDataSource's scalar foreign-key values and lets you substitute meaningful names for foreign-key values, as described in the "Substituting EntityRef for ForeignKey Values in Databound Web Controls" section of Chapter 5.

Following is a brief summary of the data types that the EntityDataSource supplies to bound controls:

- ❑ EntitySets return read/write wrapped entities with `EntityReference` key values instead of objects.
- ❑ Entity SQL queries, such as `SELECT VALUE o FROM Northwind.Orders AS o`, as the value of the `EntityDataSource.CommandText` property return entire entities that deliver read-only entities for binding.
- ❑ Entity SQL queries, such as `SELECT o.OrderID, o.OrderDate, o.ShipName FROM Northwind.Orders AS o`, as the `CommandText` property value return projection that deliver read-only `DbDataRecord` types for binding.
- ❑ Comma-separated field lists, such as `it.OrderID, it.OrderDate, it.ShipName`, as `Select` property values return read-only `DbDataRecord` types for binding.

*The `it` prefix specifies the current resultset.*

### **Adding an EntityDataSource Control to an ASP.NET Web Application Page**

Starting with a new C# or VB ASP.NET Web Application project, add a new ADO.NET Entity Data Model item named, for this example, `Northwind.edmx` with the `Northwind` sample database as its data source. Accept or change the Entity Data Model Wizard's default names — `NorthwindEntities` for the Container and Connection String — and singularize the `EntityType` names.

Drag an EntityDataSource component from the toolbox's Data section to the Design view of the Default.aspx page, open the SmartTag, and click the Configure Data Source action to start the Configure Data Source Wizard and display the Configure ObjectContext dialog. Select the Named Connection option, choose `NorthwindEntities` as the connection string, and accept `NorthwindEntities` as the DefaultContainerName (see Figure 14-5).

## Part V: Implementing the ADO.NET Entity Framework



Figure 14-5

Click Next to display the Configure Data Selection dialog, select the `EntitySet` as the `EntitySetName`, `Customer` for this example, accept the default (None) `EntityTypeFilter`, and mark the `Select All (Entity Value)` check box to return the `Customer` `EntityType` and mark the `Enable Automatic Inserts`, `Enable Automatic Updates`, and `Enable Automatic Deletes` check boxes (see Figure 14-6).



Figure 14-6

Selecting individual property values, instead of marking Select All, disables the Enable Automatic Actions check boxes and creates a read-only projection as the bound control's data source.

Click Finish to complete the configuration and add the control as EntityDataSet1 to the page.

### Exploring the Entity Datasource

The EntityDataSource shares many properties and events with the other ASP.NET 3.5 server controls that derive from the System.Web.UI.DataSourceControl class: SqlDataSource, AccessDataSource, ObjectDataSource, XmlDataSource, SiteMapDataSource, and, as mentioned earlier, LinqDataSource.

*The sample projects for this topic, EntityDataSourceWebAppCS.sln, EntityDataSourceWebAppVB.sln, LinqDataSourceWebAppCS.sln, and LinqDataSourceWebAppVB.sln are in the \Wrox\ADONET\ Chapter14 folder. The LinqDataSource examples are for comparison purposes only.*

### EntityDataSource Properties

The properties that EntityDataSource exposes are described in the following table.

Property Name	Default Value	Property Description
AutoGenerateOrderByClause*	False	Dynamically creates an Order By clause from values in the OrderByParameters collection
AutoGenerateWhereClause*	False	Dynamically creates a Where clause from values in the WhereParameters collection
AutoPage*	True	Specifies server-based data paging with a default page size of 10 entities
AutoSort*	True	Specifies server-based data sorting on the OrderBy expression unless AutoGenerateWhereClause is True
CommandText	None	An Entity SQL Query to produce a read-only resultset if EntitySet isn't specified
Connection String	name=EDMName Entities	A valid EntityClient connection string
ContextTypeName	None	A fully qualified type name for the ObjectContext
DefaultContainer	EDMNameEntities	The container name specified in the Configure Data Source wizard's first dialog
EnableDelete*	False	Enables deleting entity in databound controls
EnableInsert*	False	Enables deleting an entity in databound controls except the GridView

(continued)

## Part V: Implementing the ADO.NET Entity Framework

Property Name	Default Value	Property Description
EnableUpdate*	False	Enables editing entity property values in databound controls
EnableViewState*	True	Saves entity state as a HashTable in ViewState during postbacks
EntitySetName	EntityType	EntityType selected in the wizard's second dialog
EntityTypeFilter	None	Restricts data to the selected EntityType selected in the wizard's second dialog
GroupBy*	None	A Group By expression for aggregate queries (requires a Select expression)
Include	None	A set of comma-separated navigation paths to eager-load with the EntitySet or projection
OrderBy*	None	An Order By expression or list of parameters created in the Expression Editor
Select*	None	A set of comma-separated list of field names to return a read-only projection if EntitySet isn't specified
StoreOriginalValueInViewState*	True	Store original entity property values in ViewState (must be True if EnableUpdate or EnableDelete is True)
Where*	None	A Where expression or list of parameters created in the Expression Editor

The EntityDataSource shares property names followed by an asterisk (\*) with the LinqDataSource. The LinqDataSource fires many more events than the preceding.

The EntityDataSource applies property values that represent operations on data in the following order:

1. Where criterion for limiting the number of rows
2. Order By operator for sorting rows
3. Group By operator for aggregate data
4. Order Groups By sorts of aggregates
5. Select instruction for creating projections
6. AutoSort by user action (clicking a header)
7. AutoPage by user action (clicking a page number, first, last, previous or next button)

### Where and Group By Clauses

The Configure Data Source wizard's Configure Data Selection dialog doesn't have the LinqDataSource's options to add a `Where` constraint and `OrderBy` sort during initial configuration. If your constraint and sort expressions use literal constants, you can type them as values of the `Where` and `OrderBy` properties, such as `it.Country = "USA"` to constrain `Country` values to USA and `it.City Desc` to sort by `City` in descending order. `Where` expressions support compound constraints with `Or` and `And` operators. Separate multiple `OrderBy` expressions with commas.

Alternatively, you can use the `EntityDataSource`'s Expression Editor dialog, which is almost identical to that of the `LinqDataSource`, to add `Where`, `OrderBy`, and `Select` expressions (see Figure 14-7).

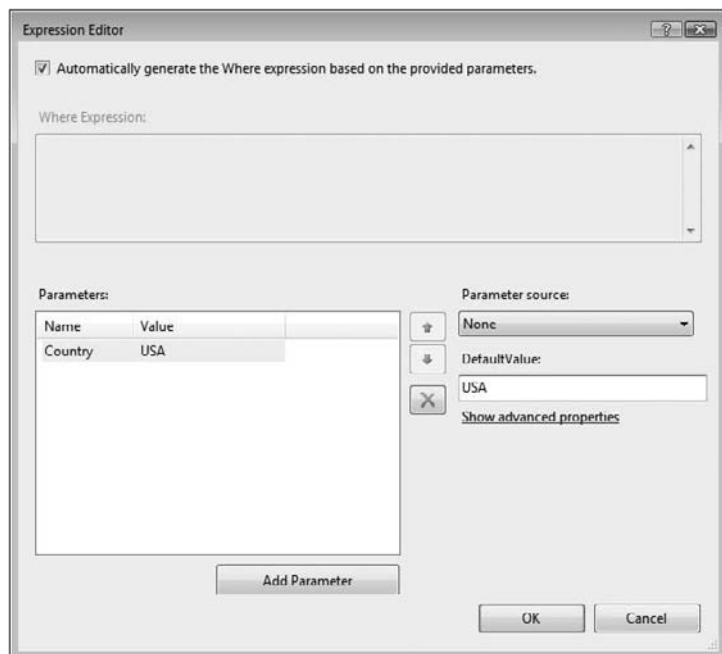


Figure 14-7

*The only significant difference between the Expression Editor's two versions is that the LinqDataSource's implementation enables you to set the GroupBy property value.*

Marking the `Automatically Generate the ... Expression` check box enables the `... Expression` text box and disables the parameter-related controls; clearing the check box does the opposite. To specify a single literal value as the `Where` expression's argument, click the `Add Parameter` button, select the `Name` text box and type the field name, `Country` for this example. Accept the default `None` as the `Parameter Source` and type the literal value, `USA` for this example, in the `Default Value` text box, as shown in Figure 14-7.

*You can't create a compound `Where` clause criterion with more than one parameter value per field; doing so causes a runtime exception. You must supply an `OrderBy` expression in this case, such as `it.Country = "USA" Or it.Country = "Canada" Or it.Country = "Mexico"`.*

## Part V: Implementing the ADO.NET Entity Framework

---

The Expression Editor also supports parameterized expressions where the parameter values are supplied at runtime by one of the Parameter Source types in the following table.

Source Type	Parameter Value Source Property
Cookie	CookieName property of a single-valued cookie
Control	ControlID (and <code>PropertyName</code> , if the default property doesn't supply the value)
Form	FormField property of an HTML form
Profile	<code>PropertyName</code> of the current user profile
QueryString	QueryString field
Session	SessionField of the Session object

Clicking the Advanced Properties link opens a properties sheet in which you can set the property values in the following table.

Property Name	Default Value	Property Description
ConvertEmptyStringToNull	True	Converts empty string values to null
DbType	Object	Lets you specify the parameter value's database data type (if the <code>Type</code> property value isn't <code>Empty</code> )
DefaultValue	Blank	Lets you specify the parameter's default value
Direction	Input	Lets you specify an <code>Output</code> , <code>InputOutput</code> , or <code>ReturnValue</code> parameter where applicable
Name	newparameter	Same as the Parameters list's Name text box
Size	0	The parameter's maximum size in characters
Type	Empty	Lets you strongly type the parameter value by selecting a CLR type, if <code>DbType</code> is empty

### EntityDataSource Events

The following table lists the events that the EntityDataSource can fire:

Event Name	Event Description
ContextCreated*	Fires after creating a new ObjectContext and provides a reference to it
ContextCreating*	Fires before creating a new ObjectContext and allows substituting a custom ObjectContext
ContextDisposing*	Fires before the ObjectContext is disposed
DataBinding*	Fires when the data-binding process commences
Deleted*	Fires after deleting a entity instance
Deleting*	Fires before deleting a entity instance
Disposed*	Fires after the EntityDataSource is disposed
Init*	Fires on page initialization
Inserted*	Fires after inserting a entity instance
Inserting*	Fires before inserting a entity instance
Load*	Fires after page loads
Prerender*	Fires before page is rendered
Selected*	Fires after Select expression executes
Selecting*	Fires before Select expression executes
Unload*	Fires after page unloads
Updated*	Fires after updating a entity instance
Updating*	Fires before updating a entity instance

*The EntityDataSource shares event names followed by an asterisk (\*) with the LinkDataSource; both controls have an identical set of property names, but the event arguments passed to event handlers and the sequence of initial events differ greatly. The EntityDataSource ... and LinkDataSource ... sample projects include event-handlers for all CustomersEDS and CustomersLDS control events.*

During page setup, EntityDataSource events fire in the following order:

1. ContextCreating
2. ContextCreated
3. Selecting
4. Selected

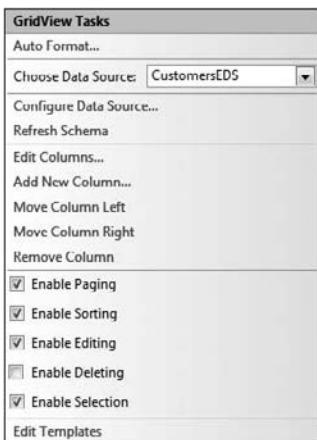
## Part V: Implementing the ADO.NET Entity Framework

---

5. Inserting, Inserted, Updating, Updated, Deleting, and Deleted, depending on update operations
6. ContextDisposing

### ***Binding and Formatting a GridView Control***

The `GridView` control is ASP.NET 3+'s primary databound server control for displaying `EntitySets` and, more commonly subsets, in a browser window. Binding a `GridView` control to the `EntityDataSource` is an extremely simple process: Drag the control to the form, open its SmartTag, and select the `EntityDataSource` you created in the preceding sections. Mark the `Enable Paging`, `Enable Sorting`, `Enable Editing`, and enable `Selection` check boxes (see Figure 14-8).



**Figure 14-8**

Optionally, click the AutoFormat link and select one named schemes, Classic for this example, open the `GridView`'s properties sheet, change its ID property value to something more descriptive, such as `CustomersGV`, open the Font property group and change the Name property value to Calibri. Finally, click the builder button in the Columns property text box to open the Fields dialog and return the columns from alphabetical order to their original sequence. With the Where property value empty, a couple of added `<div>` sections, and the Width property value set to 1500 px, the page appears with default paging properties as shown in Figure 14-9.

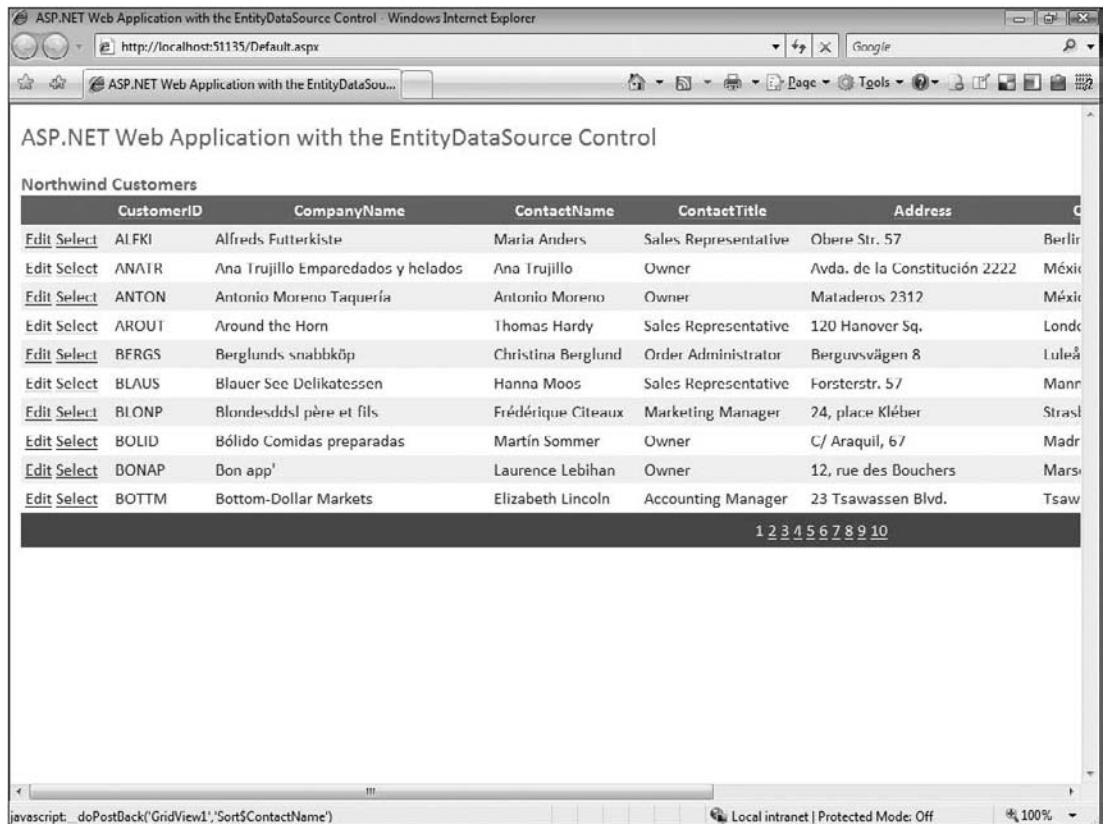


Figure 14-9

### Using the GroupBy Property and a Drop-Down List to Display Customers by Country

Paging through long customer lists isn't an efficient process, so users will appreciate the addition of a drop-down list to display results grouped by one or more criteria. The simplest approach is to add a DropDownList populated by the categories with which to group the GridView, in this case Country. After you add the DropDownList (named CountryDDL) above the GridView, add another EntityDataSource (named CustomerCountryEDS) with Customers as its EntitySet, and it. Country as its GroupBy, OrderBy, and Select property values.

*Mark the Country text box in the Configure Data Source Wizard's Configure Data Selection dialog sets the Select property value to it.Country.*

To specify the CountryDDL DropDownList's selected value as CustomerCountryEDS's Where clause criterion, open the Expression Editor for the Where property value, mark the Automatically Generate the Where Expression check box, click Add Parameter, name the parameter Country, select Control as the ParameterSource and CountryDDL as the ControlID, and type USA as the DefaultValue (see Figure 14-10).

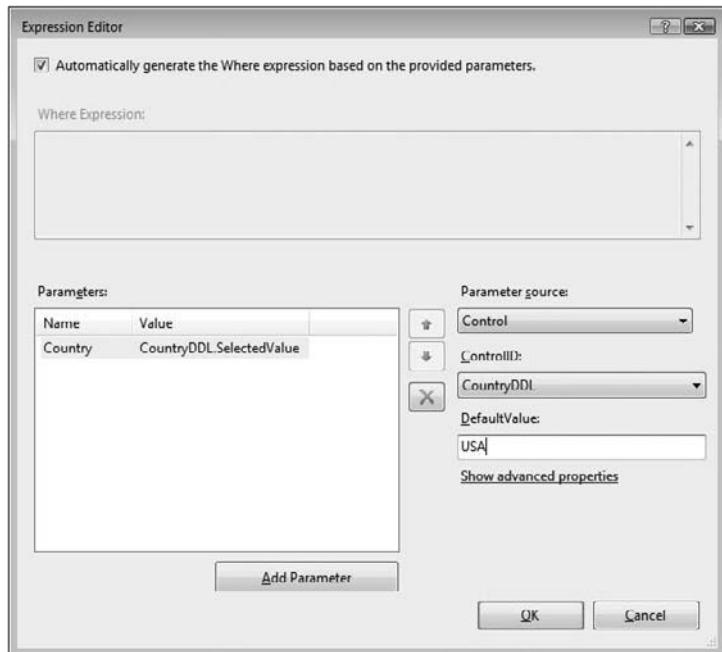


Figure 14-10

Set the CountryDDL's DataSourceID property value to CustomerCountryEDS, and the DataTextField and DataValueField property values to Country and build and run the project.

### **Adding a Linked DetailsView**

Editing entity property values in a DataGridView control can be a clumsy process when the entity has many members because a substantial amount of scrolling becomes necessary to reach lower members. Also, the DataGridView doesn't have the capability to add new entity instances. The DetailsView control is an autogenerated form that enables updating, adding, and deleting instances. The process of synchronizing a DetailsView form with a DataGridView on the same page is as simple as displaying Customer entities by their Country property.

To link a DetailsView with a GridView's selected item, start by adding a new EntityDataSource (named CustomerDetailsEDS) with Customers as its EntitySet and leave its Select All (Entity Value) check box marked.

Add a DetailsView control from the tool box, name it CustomerDV, set its Data Source to CustomerDetailsEDS, mark the Enable Editing, Enable Inserting, and Enable Deleting check boxes, and increase its width to 300 px or more. Optionally, AutoFormat the control, change its Font Family to Calibri, and use the Fields dialog to return the entity's members to their original sequence. Clear the ControlGV's Enable Editing check box because the DetailsView will handle all editing chores.

Return to the CustomerDetailsEDS, open the Expression Editor for the Where clause, mark the Automatically Generate the Where Expression check box, click Add Parameter, name the new parameter

## Chapter 14: Binding Entities to Data-Aware Controls

CustomerID, select Control as the ParameterSource and CountryDDL as the ControlID, and leave the DefaultValue empty. Optionally, change the PageSize property value to five to minimize the need to scroll the DetailView when editing in 1,024x768 resolution. Run the project, select a country, and click the Select link to display the selected instance in the DetailView control (see Figure 14-11).

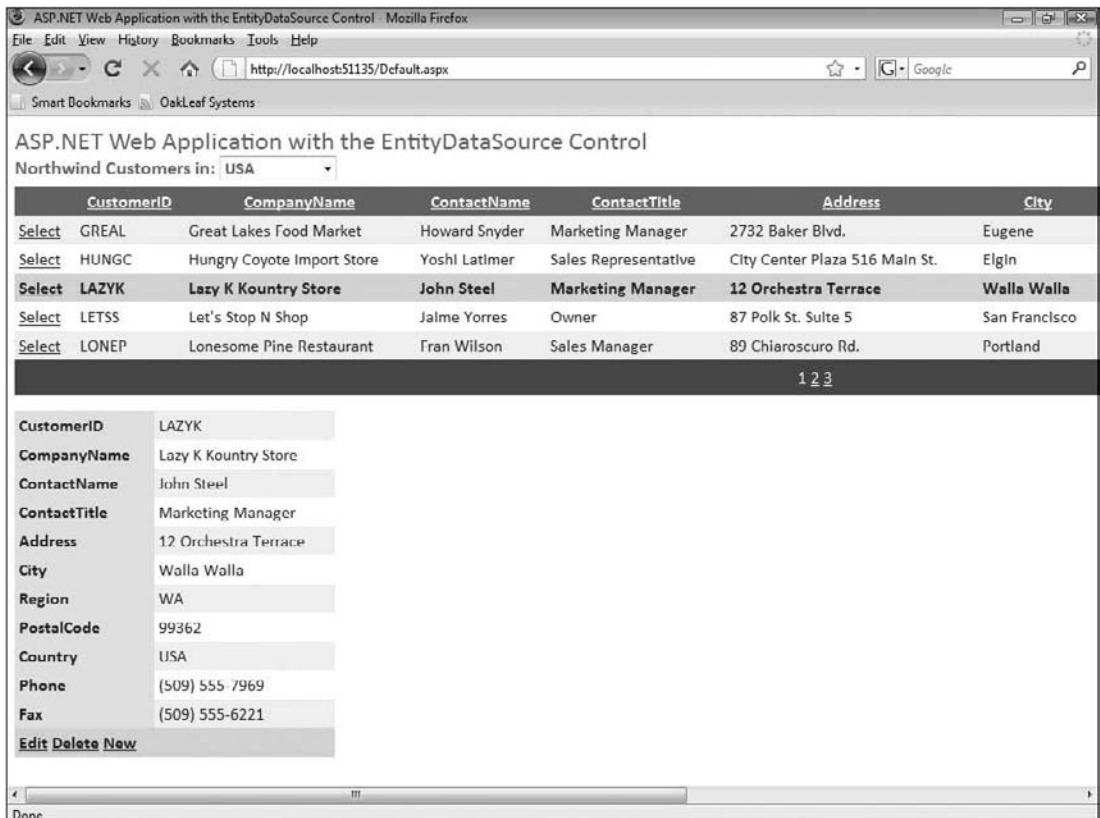


Figure 14-11

Complete the process by verifying that you can add a new Customer entity and deleting it.

## Summary

The majority of initial applications for EF v1 probably will be small-scale data entry and editing projects that take advantage of Windows and ASP.NET Web form text boxes and grids bound to *EntityObjectBindingSource* and *EntityDataSource* components. A contributor to the use of EF as a data access layer (DAL) in simple configurations is that EF shares LINQ to SQL's lack of an out-of-the-box multi-tier implementation. Both object/relational mapping tools suffer from lack of a Microsoft-supported, serializable, disconnected *ObjectContext* or *DataContext* object to track data changes on

## Part V: Implementing the ADO.NET Entity Framework

---

the client. EF v2 is expected to contain a “mini, connectionless” `ObjectContext` that corresponds to the one planned for LINQ to SQL but abandoned by the ADO.NET team.

*In the meantime, adventuresome EF developers can test drive ADO.NET architect Daniel Simmons' Perseus (a.k.a. EntityBag) project, which is available for downloading from <http://code.msdn.microsoft.com/entitybag>. Perseus uses an `EntityBag<T>` object to store an object graph with change tracking information. The CodeGallery entry includes links to seven of Simmons' detailed blog posts about the development of Perseus. You can learn more about Perseus in a “Retrieve and Update Entities across Tiers with WCF and the EntityBag” sidebar (<http://visualstudiomagazine.com/listings/list.aspx?id=292>) to the “Model Domain Objects with the Entity Framework” article from Visual Studio Magazine's March 2008 issue.*

*When this book was written, Perseus had not been updated to the VS 2008 SP1 EF version.*

The chapter's first half described wizard-generated, drag-and-drop databinding that employs techniques similar to those used with LINQ to SQL and typed `DataSet` data sources. The primary differences with EF data sources is handling changes to foreign-key members of `EntityObjects` with composite primary keys, such as the `Order_Detail` entity based on Northwind's Order Details table. Because such `EntityObjects` are immutable, either code or the user must delete the original `EntityObject` and add a new one to replace it. To maximize usability and minimize complexity, the sample projects require the user to perform the deletion before adding a new entry.

The remainder of the chapter explained binding ASP.NET Web pages with `GridView` and `FormView` server controls to `EntityDataSource` components. The `EntityDataSource` is a clone of the `LinqDataSource` for LINQ to SQL data sources but offers sorting and paging advantages.

If *n*-tier deployment is a required feature of your EF v1 project, consider using ADO.NET Data Services (formerly codenamed “Project Astoria”) with an EDM data source as its data source, as described in the next chapter. Astoria uses Representational State Transfer (REST) architecture to create Windows Communication Foundation (WCF) Web services that use JavaScript Object Notation (JSON) or the Atom Publishing Protocol (AtomPub or APP) instead of plain-old XML (POX) as the wire protocol for GET-based querying as well as inserting, updating, and deleting entities with HTTP POST, PUT, and DELETE operations. Astoria also offers a subset of the LINQ Standard Query Operators for constructing URI-based queries.

# 15

## Using the Entity Framework as a Data Source

The Entity Data Model (EDM) is the SQL Server Data Programmability (DP) group's strategic approach to data modeling and the Entity Framework (EF) is the group's first concrete implementation of EDM as a data source for .NET Frameworks. LINQ to SQL is an obvious data source alternative to EF; many developers prefer LINQ to SQL because of its better performance as well as reduced memory and processor resource consumption. However, Microsoft is making only a small ongoing investment in LINQ to SQL, so it's not likely that LINQ to SQL will gain a significantly expanded feature set in future versions.

The DP Group promotes EF as the preferred — and only updatable without modification — relational data source for the ADO.NET Data Services Framework (formerly code-named "Project Astoria" and still commonly called "Astoria") ADO.NET Data Services gained "Framework" status in early February 2008, which is very rapid promotion of a technology that was introduced less than a year earlier at Microsoft's MIX 07 conference in late April and early May 2007. References to Astoria first included the "Framework" term in a February 3, 2008 Project Astoria Team Blog post by software architect Pablo Castro, who described Astoria's objective as follows:

In general, the goal of the ADO.NET Data Services Framework is to create a simple REST-based framework for exposing data centric services. We built the framework in part from analysis of traditional websites and then looked at how architectures were changing with the move to AJAX and RIA based applications.

Astoria accepts as a read-only back-end data provider any data source that implements the IQueryble<T> interface and offers client libraries for .NET 3.5 Windows or Web form, ASP.NET Asynchronous JavaScript and XML (AJAX), and Silverlight 2 projects. To be updatable, the provider must implement the IUpdatable<T> interface in addition to IQueryble. Astoria serializes objects with the Atom Publication Protocol (AtomPub or APP) and JavaScript Object Notation

## Part V: Implementing the ADO.NET Entity Framework

---

(JSON) wire formats. Astoria's Representational State Transfer (REST) architecture enables use of HTTP POST, GET, PUT, and DELETE verbs to perform create, retrieve, update, and delete (CRUD) operations, respectively. This chapter is devoted to EF as a relational data source for Astoria Web services for Windows forms and AJAX rich Internet applications (RIAs).

*Astoria is a stopgap workaround for EF's (and LINQ to SQL's) lack of an "out-of-the box n-tier story." Astoria clients provide optional eager loading, full object graph deserialization and an elementary concurrency conflict management implementation based on HTTP ETags, which correspond approximately to timestamps.*

EF is a candidate data source for ASP.NET Dynamic Data (ANDD), a framework for quickly building complete data-driven Web sites with extended GridView, DetailsView, FormView and ListView controls by a process known to Ruby on Rails developers as *scaffolding*. Unlike Astoria, the ASP.NET team developed ANDD with LINQ to SQL as its initial data source. Projects based on EF use the EntityDataSource and those based on LINQ to SQL use the LinqDataSource; both enable CRUD operations on Web pages. However, using EF as the data source for ANDD projects requires downloading the DynamicDataEntityFrameworkWorkaround.zip file from CodePlex and modifying ANDD source code to use an alternative Microsoft.Web.DynamicData.EFProvider.dll. After installing the workaround, ANDD v1 continues to have the following problems with EF as a data source.

- Many-to-many relationships don't support navigation; a relation table is required.
- Complex types aren't supported.
- Autogenerated (int identity or rowguid) primary keys aren't detected; a partial class for the Insert page with a Scaffold value of false for the primary key field is required.
- ListView controls require special treatment of foreign-key fields.

For the preceding reasons, this chapter doesn't deal with EF as an ANDD data source.

## Creating an ADO.NET Data Services Data Source

ADO.NET Data Services is built on .NET Framework 3.5's Windows Communication Foundation (WCF) version and consists of three layers, as shown in Figure 15-1.

## Chapter 15: Using the Entity Framework as a Data Source

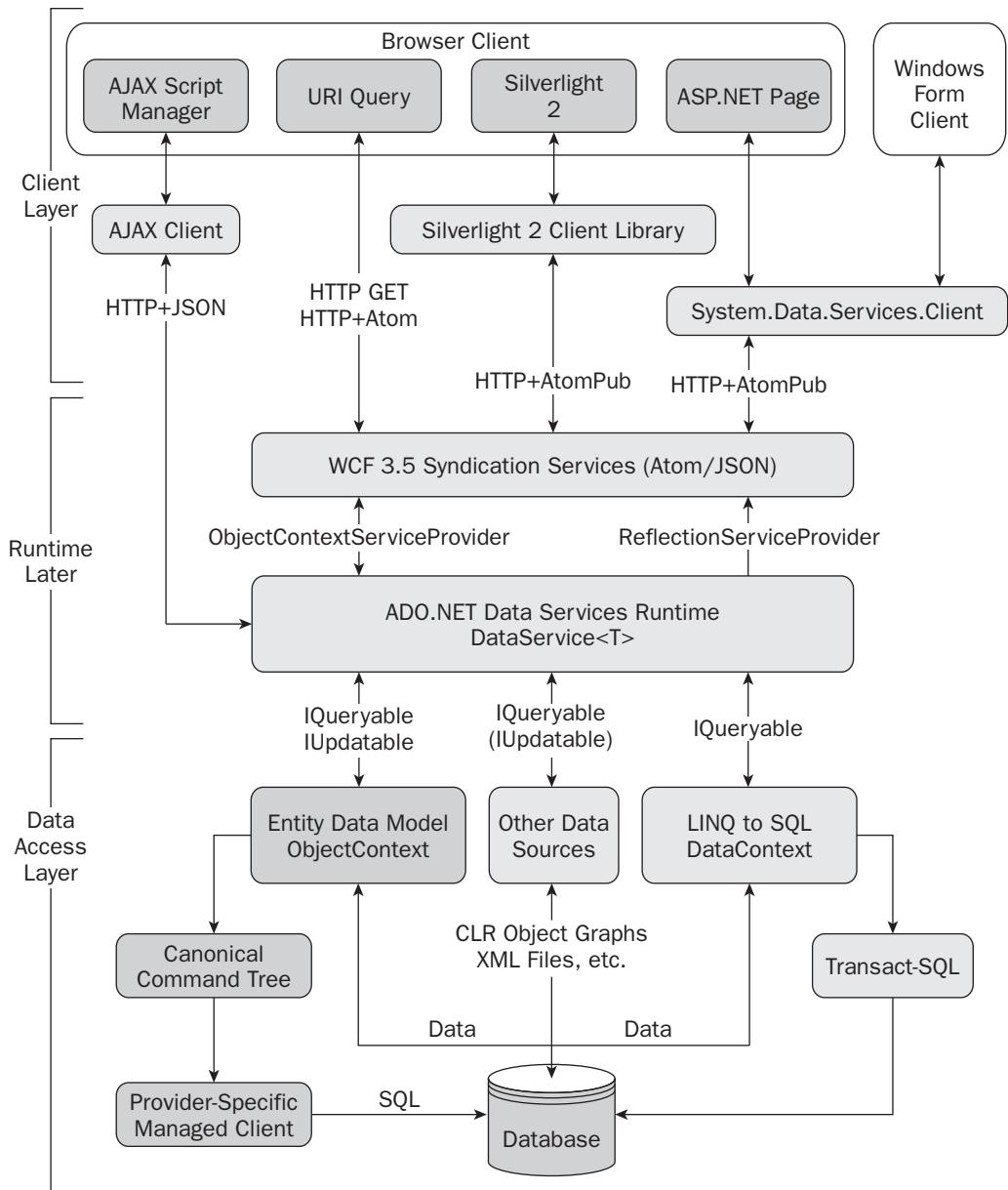


Figure 15-1

## Part V: Implementing the ADO.NET Entity Framework

---

From bottom to top, the three layers consist of:

- ❑ A pluggable data access layer (DAL), which connects to any data source that implements the `IQueryable<T>` interface. Generic type `T` must expose the equivalent of a primary key for entity instance identification. As mentioned earlier, the preferred DAL is EF, but other data sources — such as CLR object graphs (via LINQ to Objects), XML documents (via LINQ to XML), or `DataContexts` (via LINQ to SQL) — can be substituted. An internal `ReflectionServiceProvider` supports read — the HTTP `GET` verb for data sources that don't implement `IUpdatable<T>`.
- ❑ An Internet/intranet-facing runtime (`System.Data.Services`), combined with WCF syndication services, which implements Universal Resource Identifier (URI) translation, AtomPub/JSON serialized wire formats, and HTTP interaction protocols. The runtime implements the `IUpdatable<T>` interface for EF by an internal `ObjectContextServiceProvider` that enables read/write protocols with support for HTTP `GET`, `POST`, `PUT`, and `DELETE` verbs.
- ❑ A client layer (`System.Data.Services.Client`) that connects to the data services over the Internet, intranet, or both, either directly with HTTP `GET` query strings or indirectly by translating LINQ to ADO.NET Data Services (also called LINQ to REST) expressions to URIs. Specialized client libraries simplify connection to ASP.NET AJAX and Silverlight 2 clients.

## Understanding REST Architecture

Roy Fielding, one of the authors of the HTTP specification, introduced REST architecture in his doctoral dissertation, *Architectural Styles and the Design of Network-based Software Architectures* (University of California, Irvine, 1980, <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>). Chapter 5 of his dissertation describes the benefits of REST architecture:

"REST's client-server separation of concerns simplifies component implementation, reduces the complexity of connector semantics, improves the effectiveness of performance tuning, and increases the scalability of pure server components. Layered system constraints allow intermediaries — proxies, gateways, and firewalls — to be introduced at various points in the communication without changing the interfaces between components, thus allowing them to assist in communication translation or improve performance via large-scale, shared caching. REST enables intermediate processing by constraining messages to be self-descriptive: interaction is stateless between requests, standard methods and media types are used to indicate semantics and exchange information, and responses explicitly indicate cacheability."

REST identifies data items (resources) by a unique URI and accesses them by HTTP without requiring an additional messaging layer, such as SOAP. Microsoft evangelist Jon Udell describes the following "best practices for making a service 'look like the Web'" from *RESTful Web Services* (O'Reilly Media, Inc., 2007) by Leonard Richardson and Sam Ruby:

- ❑ Data are organized as sets of resources.
- ❑ Resources are addressable.
- ❑ An application presents a broad surface area of addressable resources.
- ❑ Representations of resources are densely interconnected.

As you'll see in later sections, ADO.NET Data Services satisfies all four best practices for RESTful Web services.

### **Creating a Simple Web Service and Consuming It in a Browser**

As with most other trivial implementations of ADO.NET 3.5 frameworks, you can create and demonstrate a simple EF-based Web service from pre-built VS 2008 SP1 templates in a few steps:

1. Create a new Web Application, NwindDataServicesCS.sln or NwindDataServicesVB.sln.
2. Add a new EDM of an SQL Server data source, the Northwind sample database for this example. Alternatively, add an existing EDM and copy Web.config's `<ConnectionStrings>` section from the EDM's source to the new project.
3. Add a new ADO.NET Data Service, designate the `DataContext` name, and enable read/write access to the entire data model.
4. In Solution Explorer, delete the unneeded `Default.aspx` page, right-click the `ServiceName.svc`, and choose Set as Start page.
5. Build and run the service and test it by typing URLs for traversing `EntitySets` and `EntityObjects` in the browser's address bar.

*A pair of starter projects consisting of the code for steps 1 and 2 is in the \WROX\ADONET\Chapter15\Starter\ CS\NwindDataServicesCS and ... \VB\NwindDataServicesVB folders.*

*A starter project is provided because the EDM for these projects doesn't use pluralization of `EntityType` names to `EntitySetName`. The starter projects use the Huagati DBML/EDMX Tools utility (<http://www.huagati.com/dbmltools>) to automatically correct `EntityType` and `EntitySet` names.*

Using a starter project, add a new ADO.NET Data Service to the project, rename it `Northwind.svc` for this example (see Figure 15-2), and click Add to add a `Northwind.svc.cs` or `.vb` file and several additional references to the project.

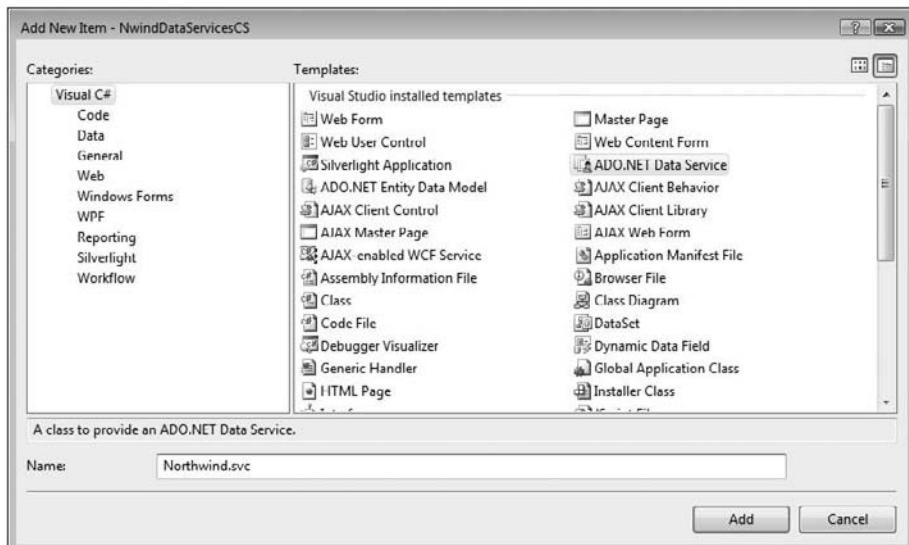


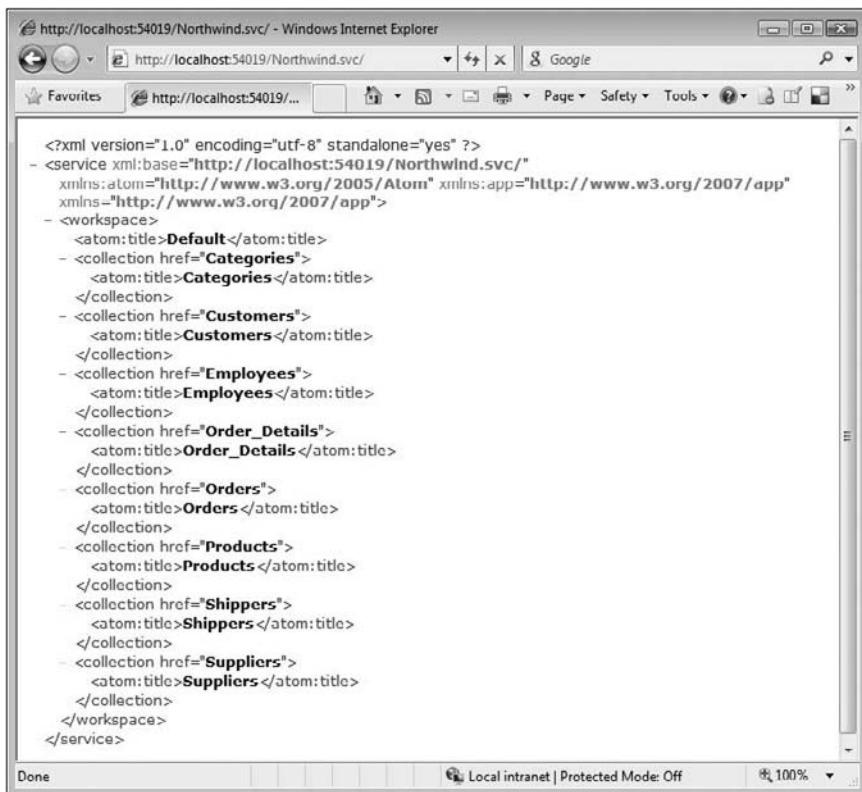
Figure 15-2

## Part V: Implementing the ADO.NET Entity Framework

Replace the TODO: put your data source class name here comment with the sample `DataContext` class name, `NorthwindEntities`.

A data source doesn't expose its resources by default, so uncomment the `config.SetEntitySetAccessRule("MyEntitySet", EntitySetRights.AllRead)` method call and temporarily replace its argument with "\*", `EntitySetRights.All` to enable read/write operations on all members of the `NorthwindModel`.

Press F5 to build and run the Web application and display *service metadata*, a list of the `NorthwindModel`'s `EntitySets` as members of the `<service>` element's default `<workspace>` in Internet Explorer, as shown for IE 8 in Figure 15-3.

A screenshot of an Internet Explorer window displaying the service metadata XML for a Northwind service. The URL in the address bar is "http://localhost:54019/Northwind.svc/". The page content shows an XML document with the following structure:

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <service xml:base="http://localhost:54019/Northwind.svc/">
  xmlns:atom="http://www.w3.org/2005/Atom" xmlns:app="http://www.w3.org/2007/app"
  xmlns="http://www.w3.org/2007/app">
  - <workspace>
    - <atom:title>Default</atom:title>
    - <collection href="Categories">
      <atom:title>Categories</atom:title>
      </collection>
    - <collection href="Customers">
      <atom:title>Customers</atom:title>
      </collection>
    - <collection href="Employees">
      <atom:title>Employees</atom:title>
      </collection>
    - <collection href="Order_Details">
      <atom:title>Order_Details</atom:title>
      </collection>
    - <collection href="Orders">
      <atom:title>Orders</atom:title>
      </collection>
    - <collection href="Products">
      <atom:title>Products</atom:title>
      </collection>
    - <collection href="Shippers">
      <atom:title>Shippers</atom:title>
      </collection>
    - <collection href="Suppliers">
      <atom:title>Suppliers</atom:title>
      </collection>
    </workspace>
  </service>
```

The XML is displayed in a monospaced font within the browser's content area.

Figure 15-3

If you've enabled Feeds in IE, you must temporarily disable Feeds on the Internet Option dialog's Content page by opening the Feeds dialog by clicking the Feeds Settings button and clearing the Turn On Feed Reading View check box.

Figure 15-3 shows that Astoria satisfies the "Data are organized as sets of resources" best practice for RESTful services criterion, because the default URI for the service metadata displays its "sets of resources" that represent all data available from the service.

## Navigating Collections and their Members

The basic format for Astoria URIs is:

```
http://host/<service>/<EntitySet>[(<Key>) [/<NavigationProperty>[(<Key>) / . . . ]]]
```

The default host URI is `localhost:port`, where `port` is the default TCP port number of ASP.NET Development Server (better known as Cassini), most commonly 54019, and `<service>` is the service name followed by the `.svc` extension, `Northwind.svc` for this chapter's examples. Astoria URIs are *case-sensitive*.

*Astoria with EF as its data source doesn't support navigation properties on derived types, which is a serious omission for developers using domain-driven design principles. The Astoria team intended to enable this capability in v1, but the fix didn't make the VS 2008 SP1 RTM date. The team plans to add the capability to v2 but doesn't offer even an approximate date for v2's RTM.*

The following URIs address the `Northwind.svc` service's individual collections and their particular members, including `EntityCollections` and `EntityReferences`:

- ❑ `http://localhost:port/Northwind.svc/Customers` returns the entire `Customers` `EntitySet` as the `<feed>` group of `<entry>` items for `Customer` `EntityObjects`, as shown in Figure 15-4. The `href` attributes of the `<link>` elements provide the syntax for obtaining an `EntityObject` specified by its `ID` and the `EntityCollection` of a one:many association.

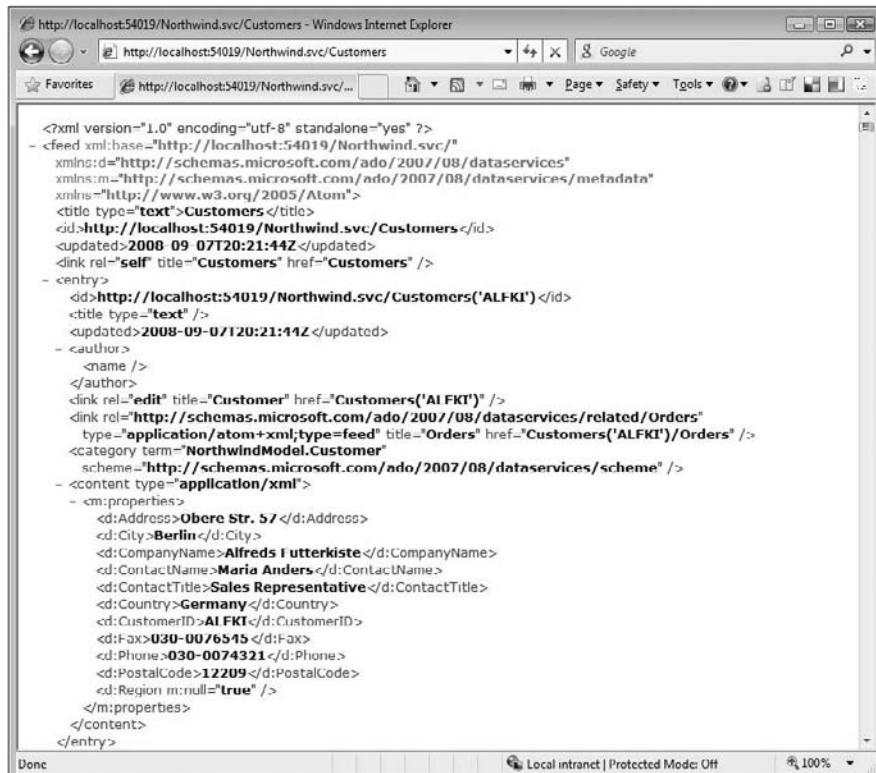
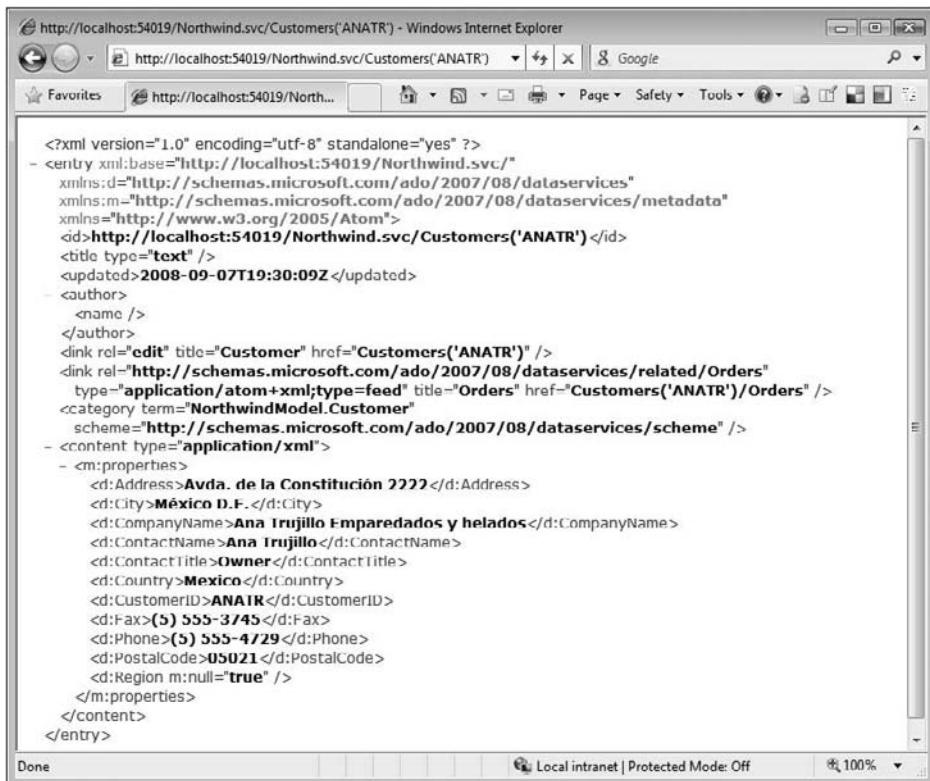


Figure 15-4

## Part V: Implementing the ADO.NET Entity Framework

- ❑ `http://localhost:port/Northwind.svc/Customers('ANATR')` returns a single `<entry>` item for the Customer entity specified by the `CustomerID` (`EntityKey`) argument of the `Customers` `EntitySet`, as shown in Figure 15-5.



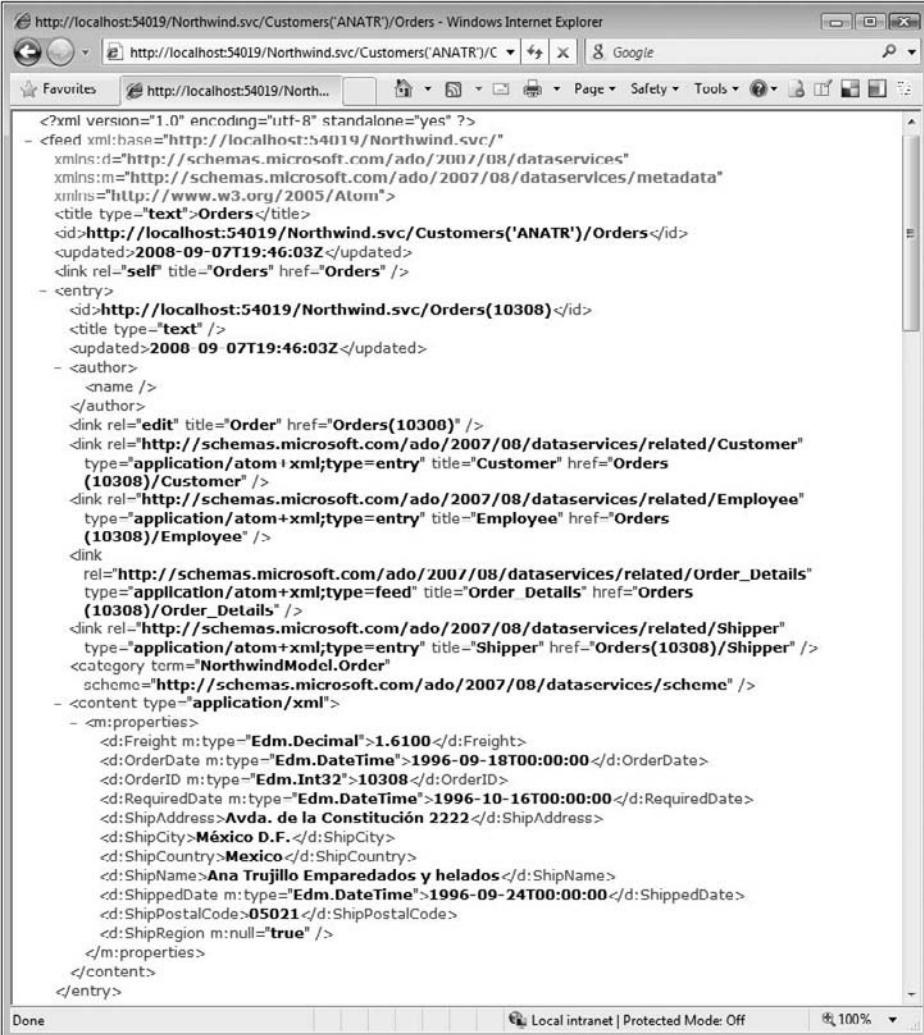
The screenshot shows a Windows Internet Explorer window displaying an XML document. The URL in the address bar is `http://localhost:54019/Northwind.svc/Customers('ANATR')`. The XML content represents a single customer entity named 'ANATR'.

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<entry xmlns:base="http://localhost:54019/Northwind.svc/">
  xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns="http://www.w3.org/2005/Atom">
    <d:uri href="http://localhost:54019/Northwind.svc/Customers('ANATR')"/>
    <id type="text" />
    <updated>2008-09-07T19:30:09Z</updated>
    <author>
      <name />
    </author>
    <link rel="edit" title="Customer" href="Customers('ANATR')"/>
    <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Orders" type="application/atom+xml;type=feed" title="Orders" href="Customers('ANATR')/Orders" />
    <category term="NorthwindModel.Customer" scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    <content type="application/xml">
      <m:properties>
        <d:Address>Avda. de la Constitución 2222</d:Address>
        <d:City>México D.F.</d:City>
        <d:CompanyName>Ana Trujillo Emparedados y helados</d:CompanyName>
        <d>ContactName>Ana Trujillo</d>ContactName>
        <d>ContactTitle>Owner</d>ContactTitle>
        <d:Country>Mexico</d:Country>
        <d:CustomerID>ANATR</d:CustomerID>
        <d:Fax>(5) 555-3745</d:Fax>
        <d:Phone>(5) 555-4729</d:Phone>
        <d:PostalCode>05021</d:PostalCode>
        <d:Region m:null="true" />
      </m:properties>
    </content>
  </entry>
```

Figure 15-5

- ❑ `http://localhost:port/Northwind.svc/Customers('ANATR')/Orders` returns a `<feed>` group of an `EntityCollection` of `Order` `<item>` elements, as shown in Figure 15-6. `<link>` elements provide the URIs of `EntityReferences` to many:one associations of `Customer`, `Employee`, and `Shipper` entities, and an `EntitySet` of `Order_Details`.

## Chapter 15: Using the Entity Framework as a Data Source



The screenshot shows a Microsoft Internet Explorer window displaying an XML document. The URL in the address bar is `http://localhost:54019/Northwind.svc/Customers('ANATR')/Orders`. The XML content represents an order with ID 10308, including its details and related entities like Customer and Shipper.

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <feed xml:base="http://localhost:54019/Northwind.svc/">
  xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns="http://www.w3.org/2005/Atom">
  <title type="text">Orders</title>
  <id>http://localhost:54019/Northwind.svc/Customers('ANATR')/Orders</id>
  <updated>2008-09-07T19:46:03Z</updated>
  <link rel="self" title="Orders" href="Orders" />
  - <entry>
    <id>http://localhost:54019/Northwind.svc/Orders(10308)</id>
    <title type="text" />
    <updated>2008-09-07T19:46:03Z</updated>
    - <author>
      <name />
    </author>
    <link rel="edit" title="Order" href="Orders(10308)" />
    <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Customer" type="application/atom+xml;type=entry" title="Customer" href="Orders(10308)/Customer" />
    <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Employee" type="application/atom+xml;type=entry" title="Employee" href="Orders(10308)/Employee" />
    <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Order_Details" type="application/atom+xml;type=feed" title="Order_Details" href="Orders(10308)/Order_Details" />
    <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Shipper" type="application/atom+xml;type=entry" title="Shipper" href="Orders(10308)/Shipper" />
    <category term="NorthwindModel.Order" scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    - <content type="application/xml">
      - <m:properties>
          <d:Freight m:type="Edm.Decimal">1.6100</d:Freight>
          <d:OrderDate m:type="Edm.DateTime">1996-09-18T00:00:00</d:OrderDate>
          <d:OrderID m:type="Edm.Int32">10308</d:OrderID>
          <d:RequiredDate m:type="Edm.DateTime">1996-10-16T00:00:00</d:RequiredDate>
          <d:ShipAddress>Avda. de la Constitución 2222</d:ShipAddress>
          <d:ShipCity>México D.F.</d:ShipCity>
          <d:ShipCountry>Mexico</d:ShipCountry>
          <d:ShipName>Ana Trujillo Emparedados y helados</d:ShipName>
          <d:ShippedDate m:type="Edm.DateTime">1996-09-24T00:00:00</d:ShippedDate>
          <d:ShipPostalCode>05021</d:ShipPostalCode>
          <d:ShipRegion m:null="true" />
        </m:properties>
      </content>
    </entry>
  </feed>
```

Figure 15-6

- ❑ `http://localhost:port/Northwind.svc/Customers('ANATR')/Orders(10308)/Shipper` returns the Shipper EntityReference of Order with an OrderID (EntityKey) value of 10308 from the Orders EntityCollection, as illustrated by Figure 15-7.

## Part V: Implementing the ADO.NET Entity Framework

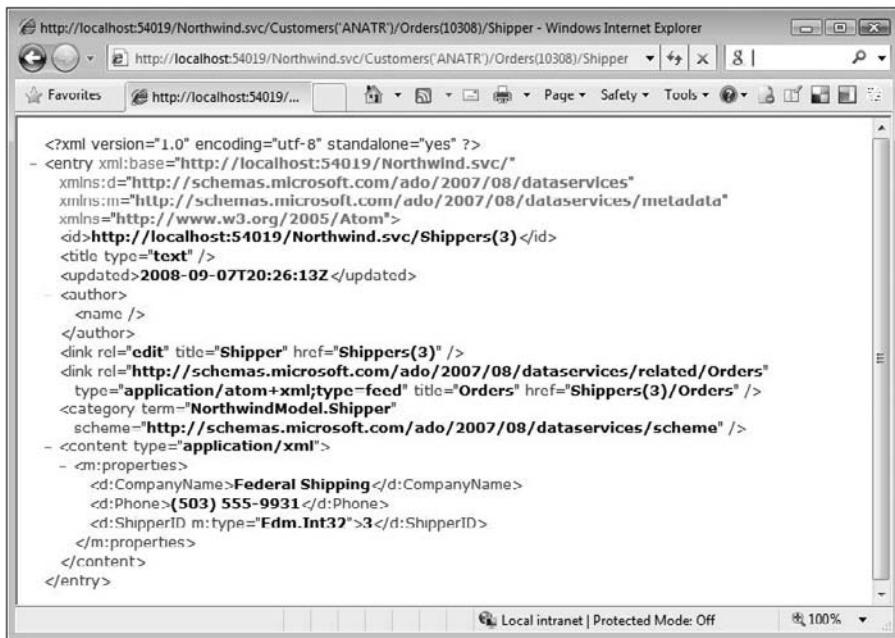


Figure 15-7

The preceding URI examples demonstrate that Astoria passes final three Web services “best practices” criteria:

- ❑ Resources are addressable.
- ❑ An application presents a broad surface area of addressable resources.
- ❑ Representations of resources are *densely interconnected*.

*The term densely interconnected is most commonly found in discussions of paralimbic cortices (archicortex and paleocortex), long-term memory, and the development of human language as a substitute for personal grooming in social groups (see Wikipedia entries for British anthropologist Robin Dunbar and Dunbar's Number). In Astoria's case, densely interconnected can be construed to refer to interconnection of addressable EntityObjects by two-way (one:many and many:one) associations.*

## Taking Advantage of Query String Options

In addition to filtering EntitySets and EntityReferences by EntityKey, Astoria supports a set of query string options for filtering and traversing object graphs. Table 15-1 describes the seven option keywords for HTTP GET operations with query strings.

*The three query string options tables that follow originally appeared in Chapter 8 and are repeated here for convenience. Astoria query strings are the same for LINQ to SQL and EF data sources.*

## Chapter 15: Using the Entity Framework as a Data Source

**Table 15-1: ADO.NET Data Services Query String Options**

Option	Description	Example
\$expand	Returns nested entities associated with the targeted entity	<pre>...Customers('ALFKI')?\$expand=Orders returns a &lt;feed&gt; document with &lt;entry&gt; elements for all Orders placed by Alfreds Futterkiste followed by a &lt;content&gt; group containing a &lt;properties&gt; group.</pre> <pre>...Customers('ALFKI')?\$expand=Orders(10643)/Employee returns an &lt;entry&gt; fragment with Order and Employee content.</pre> <pre>...Customers('ALFKI')/Orders(10262)?\$expand=Customer,Shipper,Employee returns an &lt;entry&gt; fragment with Customer, Order, Shipper, and Employee content.</pre>
\$filter	Restricts the content returned by the <i>operator</i> expression (see Table 15-2)	<pre>...Orders?\$filter=ShipCountry eq 'Brazil' returns a &lt;feed&gt; document with &lt;entry&gt; elements for all Order items shipped to Brazil.</pre> <pre>...Orders?\$filter=ShipCountry eq 'Brazil' and OrderDate gt '1996-12-31' returns all Order items shipped to Brazil after December 31, 1996.</pre>
\$orderby	Sorts the content by the specified property value ascending (asc, default) or descending (desc)	<pre>...Orders?\$orderby=ShipCountry sorts alphabetically by destination country.</pre> <pre>...Orders?\$orderby=ShipCountry desc reverses the sort order.</pre> <pre>...Orders?\$orderby=ShipCountry desc,ShipCity adds destination city to the sort.</pre>
\$skip	Skipsthe number of instances specified by the skip parameter	<pre>...Orders?\$skip=10 returns all Orders after the first 10.</pre>
\$top	Supplies the number of instances specified by the top parameter	<pre>...Orders?\$skip=10&amp;\$top=10 returns 10 Orders after the first 10.</pre> <pre>...Orders?\$skip=90&amp;\$top=10&amp;\$orderby=ShipCountry returns the tenth page with 10 Orders per page, sorted by ShipCountry.</pre>
\$metadata	Returns metadata for the data source	<pre>...\$metadata returns an &lt;edmx:Edmx&gt; document derived from the ServiceName.edmx file which includes the details of each &lt;EntityType&gt;, including elements for &lt;Key&gt;, &lt;Property&gt;, &lt;NavigationProperty&gt;, &lt;Association&gt;, and &lt;ReferentialConstraint&gt; values, as well as &lt;EntitySet&gt;, &lt;AssociationSet&gt; and, for stored procedures, &lt;FunctionImport&gt; elements.</pre>
\$value	Returns the scalar value of a property or resource	<pre>...Categories(1)/Description/\$value returns Soft drinks, coffees, teas, beers, and ales as a string (not an AtomPub document).</pre>

## Part V: Implementing the ADO.NET Entity Framework

*Ellipsis (...) in all tables' Examples column represents the standard data service URI prefix: http://localhost:port/servicename.svc/, http://localhost:54019/Northwind.svc/ for these examples.*

Table 15-2 describes query-string syntax for Astoria comparison, logical, arithmetic, and grouping operators.

**Table 15-2: ADO.NET Data Services Comparison, Logical, Arithmetic, and Grouping Operators**

Operator	Description	Example
<b>Comparison Operators</b>		
eq	Equal (=)	...Orders?\$filter=ShipCountry eq 'Brazil'
ne	Not equal (!= or <>)	...Orders?\$filter=ShipCountry ne 'Brazil'
gt	Greater than (>)	...Orders?\$filter=UnitPrice gt 50
ge	Greater than or equal (>=)	...Orders?\$filter=Freight ge 50
lt	Less than (<)	...Orders?\$filter=Freight lt 50
le	Less than or equal (<=)	...Product?\$filter=UnitPrice le 50
<b>Logical Operators</b>		
and	Logical and (&&)	...Product?\$filter=UnitPrice le 50 and UnitsInStock gt 10
or	Logical or (  )	...Product?\$filter=UnitPrice le 50 or UnitsInStock gt 10
not	Logical negation (!)	...Customers?\$filter=not endsWith(PostalCode, '50')
<b>Arithmetic Operators</b>		
add	Addition (+)	...Products?\$filter=UnitPrice add 10
sub	Subtraction (-)	...Products?\$filter=UnitPrice sub 50 gt 0
mul	Multiplication (*)	...Orders?\$filter=Freight mul 40 gt 1000
div	Division (/)	...Orders?\$filter=Freight div 10 eq 5
mod	Modulo (%)	...Orders?\$filter=Freight mod 10 eq 0
<b>Grouping Operator</b>		
( )	Precedence grouping	...Product?\$filter=(UnitPrice sub 5) gt 10

## Chapter 15: Using the Entity Framework as a Data Source

Table 15-3 lists all ADO.NET Data Services functions for DateTime and numeric data types as well as type comparison and casting.

**Table 15-3: ADO.NET Data Services Datetime, Math, and Type Functions**

DateTime Functions	String Functions
[int] day(DateTime dat0)	[string] concat([string] str0, [string] str1)
[int] hour(DateTime dat0)	[bool] contains([string] str0, [string] str1)
[int] minute(DateTime dat0)	[bool] endswith([string] str0, [string] str1)
[int] month(DateTime dat0)	[bool] startswith([string] str0, [string] str1)
[int] second(DateTime dat0)	[int] length([string] str0)
[int] year(DateTime dat0)	[int] indexof([string] str0, [string] str1)
Math Functions	
[double] round([double] num0)	[string] insert([string] str0, [int] pos, [string] str1)
[decimal] round([decimal] num0)	[string] remove([string] str0, [int] pos, [int] length)
[double] floor([double] num0)	[string] replace([string] str0, [string] find, [string] replace)
[decimal] floor([decimal] num0)	[string] sub[String]([string] str0, [int] pos)
[double] ceiling([double] num0)	[string] sub[String]([string] str0, [int] pos, [int] length)
[decimal] ceiling([decimal] num0)	[string] tolower([string] str0)
[type] Functions	
[bool] IsOf([type] typ0)	[string] toupper([string] str0)
[bool] IsOf(expression exp0, [type] typ0)	[string] trim([string] str0)
[type] typ0 Cast([type] typ0)	
[type] typ1 Cast(expression exp0, [type] typ0)	

*Data types enclosed in square brackets are for reference only; don't include explicit data types in expressions.*

## Part V: Implementing the ADO.NET Entity Framework

---

The preceding tables are primarily for reference purposes because LINQ to REST transforms conventional LINQ queries to the appropriate URI query string, as you'll see in the following sections.

### Invoking Service Operations

Service operations let you restrict the data navigation options available to end users, such as prohibiting attempts to download all orders and their order details, which generates a substantial resource hit on the service. You can restrict the scope of a query to customers from a single country, orders for a particular customer, or the like by defining a public function that executes an `ObjectQuery` to return an `IQueryable<EntityType>` collection, which can be chained with additional query options, such as `$orderby` or `$filter`. Service operation functions that invoke the HTTP GET method require decoration with a `[WebGet]` attribute.

To enable service operations, uncomment the `config.SetServiceOperationAccessRule()` method call and temporarily change the "MyServiceOperation" argument value to "\*".

As examples, following are `CustomersByCountry` functions that return an unsorted `IQueryable<Customer>` type for a valid `Customer.Country` property value by executing an `ObjectQuery` a Query Builder Method (QBM):

#### C# 3.0

```
[WebGet]
public IQueryable<Customer> CustomersByCountry(string country)
{
    if (string.IsNullOrEmpty(country))
        throw new ArgumentNullException("country",
            "You must provide a 'Country' argument value.");
    return this.CurrentDataSource.Customers.Where("it.Country = @country",
        new ObjectParameter("country", country));
}
```

#### VB 9.0

```
<WebGet> _
Public Function CustomersByCountry(ByVal country As String) As IQueryable(Of
Customer)
    If String.IsNullOrEmpty(country) Then
        Throw New ArgumentNullException("country",
            "You must provide a 'Country' argument value.")
    End If
    Return Me.CurrentDataSource.Customers.Where("it.Country = @country",
        New ObjectParameter("country", country))
End Function
```

Executing the following URI returns a `<feed>` document with `<entry>` elements for each Brazilian Customer entity sorted by `City` property values:

```
http://localhost:54019/Northwind.svc/
    GetCustomersByCountry?country='Brazil'&$orderby=City
```

## Chapter 15: Using the Entity Framework as a Data Source

---

The “Composing Query Builder Methods to Write ObjectQueries” section of Chapter 12 provides more information of QBMs.

A more interesting ObjectQuery composed by QBM returns the last five orders for a Customer specified by the CustomerID property value:

### C# 3.0

```
[WebGet]
public IQueryable<Order> GetLastFiveOrdersByCustomerID(string id)
{
    if (string.IsNullOrEmpty(id))
        throw new ArgumentNullException("id",
            "You must provide a 'CustomerID' argument value.");
    return this.CurrentDataSource.Orders.Where("it.Customer.CustomerID = @id",
        new ObjectParameter("id", id)).OrderBy("it.OrderID DESC").Top("5");
}
```

### VB 9.0

```
Public Function GetLastFiveOrdersByCustomerID(ByVal id As String) As IQueryable(Of Order)
    If String.IsNullOrEmpty(id) Then
        Throw New ArgumentNullException("id",
            "You must provide a 'CustomerID' argument value.")
    End If
    Return Me.CurrentDataSource.Orders.Where("it.Customer.CustomerID = @id",
        New ObjectParameter("id", id)).OrderBy("it.OrderID DESC").Top("5")
End Function
```

The following URL returns the last five orders for Alfreds Futterkiste:

```
http://localhost:54019/Northwind.svc/GetLastFiveOrdersByCustomerID?id='ALFKI'
```

## Consuming ADO.NET Data Services with the .NET 3.5 Client Library

The .NET Framework 3.5 Client Library (`System.Data.Services.Client` namespace) for Windows form and ASP.NET Web form projects provides a top-level `DataServiceContext` object that corresponds to EF's `ObjectContext` or LINQ to SQL's `DataContext` object. A separate ADO.NET Data Services AJAX Client Library supports RIAs, as described in the later section “Consuming ADO.NET Data Services with the AJAX Client Library.” There's also a client library for Silverlight 2.0; Silverlight clients are beyond the scope of this book.

## Part V: Implementing the ADO.NET Entity Framework

The Client Library enables Windows or Web form clients to incorporate an autogenerated object model for specific types of AtomPub resources. The client's `DataServiceQuery()` method executes LINQ to REST queries that return strongly typed `IQueryable<T>` sequences. The library also simplifies data binding and update semantics by treating resources with associations (relationships) as members of an object graph.

Autogenerating the data service classes of a WinForm client for an ADO.NET Data Service involves the following steps:

1. Open the service project's Properties window, select the Web page, fix the TCP port by selecting the Specific Port option, 54019 for this example, close the Properties window, and press F5 to build and run the service.
2. Create a new Windows Form client project, NwindAdoNetDataServicesClient, and add a reference to the `System.Data.Services.Client` namespace.
3. Add a Service Reference to open the Add a Service Reference dialog, type the service address — `http://localhost:54019/Northwind.svc` for this example — in the Address text box and give the proxy a descriptive namespace (`NwindDataService`, see Figure 15-8.) Click OK to create the client proxy.

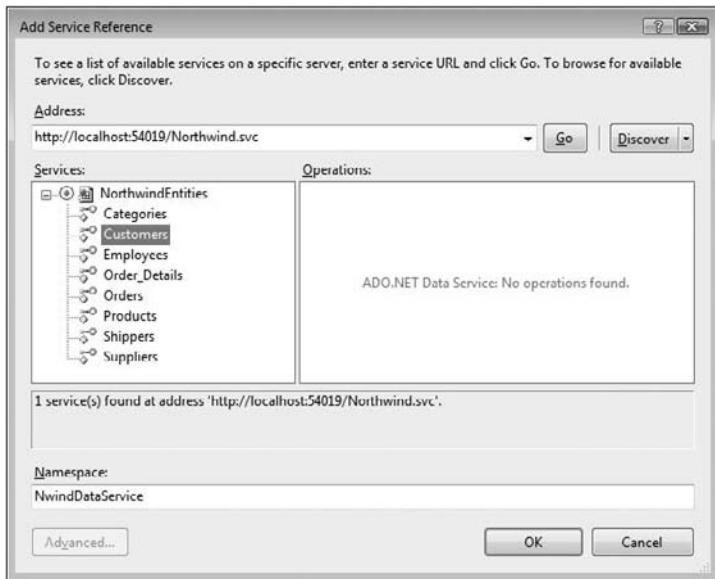


Figure 15-8

If the solution contains the client and service projects, you can click the Add a Service Reference dialog's Discover button to obtain the service address automatically. In this case, the service doesn't need to be running.

## Chapter 15: Using the Entity Framework as a Data Source

---

The NwindDataService proxy consists of the Reference.datasvcmap, Reference.cs or Reference.vb, and service.edmx files. Reference.datasvcmap is an XML file that specifies the locations and IDs of related files:

### XML (Reference.datasvcmap)

```
<?xml version="1.0" encoding="utf-8"?>
<ReferenceGroup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    ID="e489b1e6-7ef0-469e-b151-2ff87f60858b"
    xmlns="urn:schemas-microsoft-com:xml-dataservicemap">
    <MetadataSources>
        <MetadataSource Address="http://localhost:54019/Northwind.svc"
            Protocol="http" SourceId="1" />
    </MetadataSources>
    <Metadata>
        <MetadataFile FileName="service.edmx" MetadataType="Edmx"
            ID="94df2fdd-72a9-4bcf-8d98-431c795d8f05" SourceId="1"
            SourceUrl="http://localhost:54019/Northwind.svc" />
    </Metadata>
    <Extensions />
</ReferenceGroup>
```

Reference.cs or Reference.vb contains the NorthwindEntities class, which implements DataServiceContext and DataServiceQuery objects to return the eight exposed EntitySets.

*The NwindAdoNetDataServicesClientCS.sln and ...VB.sln sample projects are located in the \WROX\ADONET\Chapter15\CS and ...VB folders.*

## Executing Queries from Windows Form Clients

The following sections provide examples of these types of Astoria queries that return sequences:

- Query-string URIs
- DataServiceQuery
- LINQ to REST queries
- Queries that return associated entities
- Batch Queries with DataServiceRequests

### Executing a Query-String URI from a Client

The .NET Framework 3.5's service proxy instance of the DataServiceContext.Execute<T> is the method for passing query strings as a relative URI for execution by the service. The proxy instance of the NorthwindEntities type has the same Absolute URI prefix as browser-based queries. A Relative URI suffix contains query-string options, operators, and functions to complete the query expression.

## Part V: Implementing the ADO.NET Entity Framework

---

The following event-handlers, with UriKind enumerations emphasized, return an AtomPub <feed> group with <item> elements for Customers located in Brazil sorted by City name:

### C# 3.0

```
private void btnBrazilianCustomersQS_Click(object sender, EventArgs e)
{
    string query = "Customers?$filter=Country eq 'Brazil'&$orderby=City";
    NorthwindEntities proxy = new NorthwindEntities(new
        Uri("http://localhost.:54019/Northwind.svc", UriKind.Absolute));
    proxy.MergeOption = MergeOption.AppendOnly;

    IEnumerable<Customer> customers = proxy.Execute<Customer>(
        new Uri(query, UriKind.Relative));
    foreach (Customer c in customers)
    {
        txtResult.Text += c.CompanyName + ", " + c.City + "\r\n";
    }
}
```

### VB 9.0

```
Private Sub btnBrazilianCustomersQS_Click(ByVal sender As Object, _
    ByVal e As EventArgs)
    Dim query As String = "Customers?$filter=Country eq 'Brazil'&$orderby=City"
    Dim proxy As New NorthwindEntities(New _
        Uri("http://localhost.:54019/Northwind.svc", UriKind.Absolute))
    proxy.MergeOption = MergeOption.AppendOnly

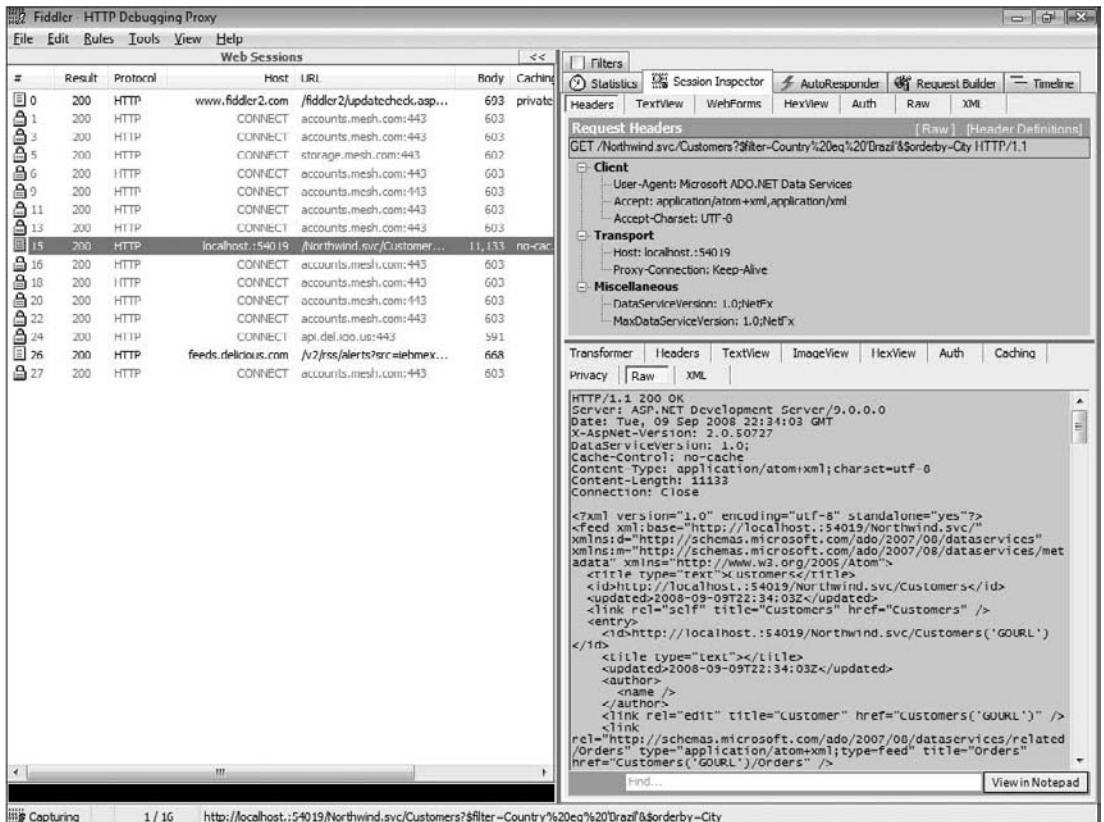
    Dim customers As IEnumerable(Of Customer) = _
        proxy.Execute(Of Customer)(New Uri(query, UriKind.Relative))

    ' Iterate the result
    For Each c As Customer In customers
        txtResult.Text += c.CompanyName & ", " & c.City & vbCrLf
    Next c
End Sub
```

*The query-string URI approach to returning AtomPub feed documents is the fastest of the three methods described in this and the following two sections. Best performance is to be expected because the alternatives don't offer filters or require translating the LINQ query to the appropriate query-string URI. Code for displaying elapsed time has been removed from the examples for brevity.*

The emphasized period (.) suffix to localhost enables the Fiddler2 Web Debugger tool to capture traffic from the 127.0.0.1 (localhost) address without affecting connection to the data Web service. Fiddler2 is a useful Microsoft utility to capture HTTP data on the wire. Figure 15-9 shows Fiddler2 displaying the HTTP Request Headers and payload for the preceding query.

Chapter 15: Using the Entity Framework as a Data Source



**Figure 15-9**

You can download and install Fiddler2, which runs as an IE7+ add-in, from <http://fiddler2.com>.

## **Executing a DataServiceQuery**

The `DataServiceQuery` approach is of limited utility because it's restricted to returning an entire `EntitySet`. If you attempt to add a `$filter`, `$orderby`, or other option to the query expression, you receive an "Expected a relative URL path without query or fragment" exception.

## Part V: Implementing the ADO.NET Entity Framework

---

Following is a sample DataServiceQuery with post-processing by the client:

### C# 3.0

```
private void btnBrazilianCustomersDS_Click(object sender, EventArgs e)
{
    NorthwindEntities proxy = new NorthwindEntities(new
        Uri("http://localhost.:54019/Northwind.svc", UriKind.Absolute));
    proxy.MergeOption = MergeOption.AppendOnly;
    IQueryable<Customer> customers = proxy.CreateQuery<Customer>("Customers");

    foreach (Customer c in customers)
    {
        if (c.Country == "Brazil")
            txtResult.Text += c.CompanyName + ", " + c.City + "\r\n";
    }
}
```

### VB 9.0

```
Private Sub btnBrazilianCustomersDS_Click(ByVal sender As Object, ByVal e As
EventArgs)
    Dim proxy As New NorthwindEntities(New _
        Uri("http://localhost.:54019/Northwind.svc", UriKind.Absolute))
    proxy.MergeOption = MergeOption.AppendOnly
    Dim customers As IEnumerable(Of Customer) = _
        proxy.CreateQuery(Of Customer)("Customers")

    For Each c As Customer In customers
        If c.Country = "Brazil" Then
            txtResult.Text += c.CompanyName & ", " & c.City & Constants.vbCrLf
        End If
    Next c
End Sub
```

The `CreateQuery()` method returns an `IQueryable<T>` sequence that you can filter on the client, but it's a good practice to avoid throwing away items by using one of the other two query methods.

## **Executing LINQ to REST Queries**

Invoking the `Execute()` method with a URI query string gives the best performance, but most developers probably will opt for LINQ to REST queries because of the more familiar LINQ syntax. LINQ to REST has far fewer valid Standard Query Operators (SQOs) than LINQ to Entities or LINQ to SQL.

### C# 3.0

```
private void btnBrazilianCustomersLINQ_Click(object sender, EventArgs e)
{
    NorthwindEntities proxy = new NorthwindEntities(new
        Uri("http://localhost.:54019/Northwind.svc", UriKind.Absolute));
    proxy.MergeOption = MergeOption.AppendOnly;

    var customers = from c in proxy.Customers
                    where c.Country == "Brazil"
                    orderby c.City descending
```

```
        select c;
foreach (Customer c in customers)
{
    txtResult.Text += c.CompanyName + ", " + c.City + "\r\n";
}
}
```

## VB 9.0

```
Private Sub btnBrazilianCustomersLINQ_Click(ByVal sender As Object, _
    ByVal e As EventArgs)
    Dim proxy As New NorthwindEntities(New _
        Uri("http://localhost.:54019/Northwind.svc", UriKind.Absolute))
    proxy.MergeOption = MergeOption.AppendOnly

    Dim customers = From c In proxy.Customers _
        Where c.Country = "Brazil" _
        Order By c.City Descending _
        Select c
    For Each c As Customer In customers
        txtResult.Text += c.CompanyName & ", " & c.City & Constants.vbCrLf
    Next c
End Sub
```

DataServiceQuery and LINQ to REST queries throw an exception when iterating an empty sequence because they don't respect the `FirstOrDefault()` SQO. Thus, the "Changing EntitySets with URI Queries and DataService.SaveChanges()" sections' examples use query-string URIs to test for the presence or absence of an item.

## Returning Associated Entities

The simplest way to return associated entries nested within their parent entry is to translate a LINQ query to an absolute URI, append an `&$expand=AssociatedEntity`, and replace the proxy's original URI, as shown for the Orders and their Customers query of the following event handler:

## C# 3.0

```
// Retrieve the last five Orders from Brazil with associated Customer objects
private void btnCustomersOrdersAssoc_Click(object sender, EventArgs e)
{
    NorthwindEntities proxy = new NorthwindEntities(new
        Uri("http://localhost.:54019/Northwind.svc", UriKind.Absolute));
    proxy.MergeOption = MergeOption.AppendOnly;

    // Compose the URI query and append $expand
    var orders = (from o in proxy.Orders
                  where o.Customer.Country == "Brazil"
                  orderby o.OrderID descending
                  select o).Take(5);
    string ordersUri = orders.ToString();
    ordersUri += "&$expand=Customer";

    // Execute the URI query
```

## Part V: Implementing the ADO.NET Entity Framework

---

```
IEnumerable<Order> ords = proxy.Execute<Order>(
    new Uri(ordersUri, UriKind.Absolute));

txtResult.Text =
    "Last five Orders from Brazil with associated Customer objects\r\n\r\n";
foreach (Order o in ords)
{
    txtResult.Text += string.Format("OrderID = {0},",
        OrderDate = {1}\r\n", o.OrderID, o.OrderDate);
    txtResult.Text += string.Format("CustomerID = {0}, CompanyName =
        {1}\r\n\r\n", o.Customer.CustomerID, o.Customer.CompanyName);
}
}
```

### VB 9.0

```
' Retrieve the last five Orders from Brazil with associated Customer objects
Private Sub btnCustomersOrdersAssoc_Click(ByVal sender As Object, _
    ByVal e As EventArgs)
    Dim proxy As New NorthwindEntities(New
        Uri("http://localhost:54019/Northwind.svc", UriKind.Absolute))
    proxy.MergeOption = MergeOption.AppendOnly

    ' Compose the URI query and append $expand
    Dim orders = (From o In proxy.Orders _
        Where o.Customer.Country = "Brazil" _
        Order By o.OrderID Descending _
        Select o).Take(5)
    Dim ordersUri As String = orders.ToString()
    ordersUri &= "&$expand=Customer"

    ' Execute the URI query
    Dim ords As IEnumerable(Of Order) = _
        proxy.Execute(Of Order)(New Uri(ordersUri, UriKind.Absolute))

    txtResult.Text = "Last five Orders from Brazil with associated " & _
        "Customer objects" & vbCrLf & vbCrLf
    For Each o As Order In ords
        txtResult.Text += String.Format("OrderID = {0}, OrderDate = {1}", _
            o.OrderID, o.OrderDate) & vbCrLf
        txtResult.Text += String.Format("CustomerID = {0}, CompanyName = {1} ", _
            o.Customer.CustomerID, o.Customer.CompanyName) & vbCrLf & vbCrLf
    Next o
End Sub
```

Figure 15-10 shows the NwindAdoNetDataServicesClientCS.sln project after executing the preceding query.

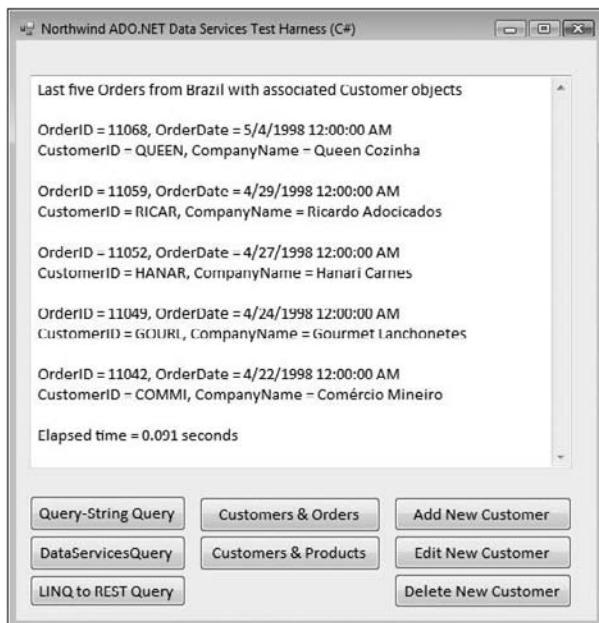


Figure 15-10

Depending upon your domain model and the number of items returned, it might be more efficient to prevent downloading duplicate associated items by downloading both `EntitySets` in a batch, saving them to `IList<T>` collections, and executing a local LINQ to Object join query to emulate the associations. Alternatively, use the `DataContext.SetLink(SourceObject, "SourcePropertyName", TargetObject)` method to establish a link between `EntityObjects` in a many:1 association, such as `Order:Customer`.

### **Executing Batch Queries with DataServiceRequests**

Collections of `DataServiceRequest<T>(new Uri("QueryUri1"))` instances, together with the `DataServiceResponse.ExecuteBatch(DataServiceRequest[])` method perform batch queries.

The following event handler generates URIs for `Customer` and `Product` queries by applying the `ToString()` method to the LINQ to REST queries, creates a `batchResponses` collection of the `DataServiceResponse` type, casts the `DataServiceResponse` as the `QueryOperationResponse` type (emphasized), and then filters the `QueryOperationResponse` with its `OfType<GenericType>` method (also emphasized).

## Part V: Implementing the ADO.NET Entity Framework

---

### C# 3.0

```
private void btnCustsProdsBatch_Click(object sender, EventArgs e)
{
    NorthwindEntities proxy = new NorthwindEntities(new
        Uri("http://localhost.:54019/Northwind.svc", UriKind.Absolute));
    proxy.MergeOption = MergeOption.AppendOnly;

    var customers = (from c in proxy.Customers
                      where c.Country == "Brazil"
                      orderby c.City
                      select c).Take(5);
    Uri uriCusts = new Uri(customers.ToString());

    var products = (from p in proxy.Products
                    where p.Category.CategoryID == 1
                    orderby p.ProductName
                    select p).Take(5);
    Uri uriProds = new Uri(products.ToString());

    DataServiceResponse batchResponses =
        proxy.ExecuteBatch(new DataServiceRequest<Customer>(uriCusts),
                           new DataServiceRequest<Product>(uriProds));

    txtResult.Text =
        "Batched Query Demonstration with Customers and Products EntitySets\r\n\r\n";
    bool isCustomer = true;
    foreach (var response in batchResponses)
    {
        // Cast DataServiceResponse as QueryOperationResponse
        QueryOperationResponse queryOpResp = response as QueryOperationResponse;
        if (isCustomer)
        {
            txtResult.Text += "First five Customers in Brazil by City\r\n";
            foreach (var c in queryOpResp.OfType<Customer>())
                txtResult.Text += "    " + c.CompanyName + ", " + c.City + "\r\n";
            isCustomer = false;
        }
        else
        {
            txtResult.Text +=
                "\r\nFirst five Products in Beverages Category by ProductName\r\n";
            foreach (var p in queryOpResp.OfType<Product>())
                txtResult.Text += "    " + p.ProductName + ", " +
                    p.QuantityPerUnit + "\r\n";
        }
    }
}
```

## Chapter 15: Using the Entity Framework as a Data Source

---

### VB 9.0

```
Private Sub btnCustsProdsBatch_Click(ByVal sender As Object, ByVal e As EventArgs)
    Dim proxy As New NorthwindEntities(New
        Uri("http://localhost.:54019/Northwind.svc", UriKind.Absolute))
    proxy.MergeOption = MergeOption.AppendOnly

    Dim customers = (From c In proxy.Customers _
                      Where c.Country = "Brazil" _
                      Order By c.City _
                      Select c).Take(5)
    Dim uriCusts As New Uri(customers.ToString())

    Dim products = (From p In proxy.Products _
                    Where p.Category.CategoryID = 1 _
                    Order By p.ProductName _
                    Select p).Take(5)
    Dim uriProds As New Uri(products.ToString())

    Dim batchResponses As DataServiceResponse = _
        proxy.ExecuteBatch(New DataServiceRequest(Of Customer)(uriCusts), _
                           New DataServiceRequest(Of Product)(uriProds))

    txtResult.Text = _
        "Batched Query Demonstration with Customers and Products EntitySets" & _
        vbCrLf & vbCrLf
    Dim isCustomer As Boolean = True
    For Each response In batchResponses
        ' Cast DataServiceResponse as QueryOperationResponse
        Dim queryOpResp As QueryOperationResponse = _
            TryCast(response, QueryOperationResponse)
        If isCustomer Then
            txtResult.Text &= _
                "First five Customers in Brazil by City" & vbCrLf
            For Each c In queryOpResp.OfType(Of Customer)()
                txtResult.Text &= "    " & c.CompanyName & ", " & c.City & vbCrLf
            Next c
            isCustomer = False
        Else
            txtResult.Text += vbCrLf & _
                "First five Products in Beverages Category by ProductName" & vbCrLf
            For Each p In queryOpResp.OfType(Of Product)()
                txtResult.Text &= "    " & p.ProductName & ", " & _
                    p.QuantityPerUnit & vbCrLf
            Next p
        End If
    Next response
End Sub
```

*When this book was written, there was almost no correct documentation for or demonstrations of batch queries. The lack of resources probably is due to significant changes to VS 2008 SP1 between the beta 2 and release versions.*

### **Changing EntitySets with URI Queries and DataService.SaveChanges()**

The following sections show you how to add, update, and delete single entities, manage optimistic concurrency conflicts, and batch a `ChangeSet` with changes to multiple `EntityObjects`.

#### **Adding New Entities**

To add a new `EntityObject` to an `EntitySet`, create a new entity, populate its properties, create the proxy, invoke the `proxy.AddToEntitySet(NewEntityObject)` method, and apply the `ApplyChanges()` method. The optional `dsrAdd` `DataServiceResponse` object lets you inspect the `HttpResponse` for the insert's outcome. Testing for conflicts prior to addition and success after addition is optional.

The following sample event handler adds a BOGUS customer with “before and after” tests:

#### **C# 3.0**

```
// Add new Bogus Software, Inc. Customer
private void btnAddCust_Click(object sender, EventArgs e)
{
    Customer custBogus = new Customer();
    custBogus.CustomerID = "BOGUS";
    custBogus.CompanyName = "Bogus Software, Inc.";
    custBogus.ContactName = "Joe Bogus";
    custBogus.Address = "1221 Broadway";
    custBogus.City = "Oakland";
    custBogus.Region = "CA";
    custBogus.PostalCode = "94612";
    custBogus.Country = "USA";
    custBogus.Phone = "(510) 555-1212";
    custBogus.Fax = "(510) 555-1213";

    // Generate the proxy, test, add the new Customer, SaveChanges, and test
    NorthwindEntities proxy = new NorthwindEntities(new
        Uri("http://localhost.:54019/Northwind.svc", UriKind.Absolute));

    // Test for presence of BOGUS Customer with a query-string URI
    string query = "Customers?$filter=CustomerID eq 'BOGUS'";
    List<Customer> bogus = proxy.Execute<Customer>(
        new Uri(query, UriKind.Relative)).ToList();
    if (bogus.Count() > 0)
    {
        txtResult.Text = "BOGUS Customer already added.";
        return;
    }

    // Add BOGUS Customer
    proxy.AddToCustomers(custBogus);
    DataServiceResponse dsrAdd = null;
```

```
dsrAdd = proxy.SaveChanges();

// Verify added
IEnumerable<Customer> bogus2 = proxy.Execute<Customer>(
    new Uri(query, UriKind.Relative));
foreach (var b in bogus2)
{
    txtResult.Text += "Added new Customer with: \r\n" +
        "CustomerID = " + b.CustomerID + "\r\n" +
        "CompanyName = " + b.CompanyName + "\r\n" +
        "ContactName = " + b.ContactName + "\r\n";
    return;
}
txtResult.Text = "Failed to add BOGUS Customer.";
}
```

### VB 9.0

```
' Add new Bogus Software, Inc. Customer
Private Sub btnAddCust_Click(ByVal sender As Object, ByVal e As EventArgs)
    Dim custBogus As New Customer()
    With custBogus
        .CustomerID = "BOGUS"
        .CompanyName = "Bogus Software, Inc."
        .ContactName = "Joe Bogus"
        .Address = "1221 Broadway"
        .City = "Oakland"
        .Region = "CA"
        .PostalCode = "94612"
        .Country = "USA"
        .Phone = "(510) 555-1212"
        .Fax = "(510) 555-1213"
    End With

    ' Generate the proxy, test, add the new Customer, SaveChanges, and test
    Dim proxy As New NorthwindEntities(New _
        Uri("http://localhost.:54019/Northwind.svc", UriKind.Absolute))

    ' Test for presence of BOGUS Customer with a query-string URI
    Dim query As String = "Customers?$filter=CustomerID eq 'BOGUS'"
    Dim bogus As List(Of Customer) = _
        proxy.Execute(Of Customer)(New Uri(query, UriKind.Relative)).ToList()
    If bogus.Count() > 0 Then
        txtResult.Text = "BOGUS Customer already added."
        Return
    End If

    ' Add BOGUS Customer
    proxy.AddToCustomers(custBogus)
    Dim dsrAdd As DataServiceResponse = Nothing
    dsrAdd = proxy.SaveChanges()

    ' Verify added
```

## Part V: Implementing the ADO.NET Entity Framework

---

```
Dim bogus2 As IEnumerable(Of Customer) = _
    proxy.Execute(Of Customer)(New Uri(query, UriKind.Relative))
For Each b In bogus2
    txtResult.Text &= "Added new Customer with: " & vbCrLf & _
        "CustomerID = " & b.CustomerID & vbCrLf & _
        "CompanyName = " & b.CompanyName & vbCrLf & _
        "ContactName = " & b.ContactName & vbCrLf
    Return
Next b
txtResult.Text = "Failed to add BOGUS Customer."
End Sub
```

### Updating Entities

Updating entities follows the pattern for updating EF EntitySets directly: Modify the client-side `EntityObject` instance, invoke the `proxy.UpdateObject(EntityObject)` method, and apply the `ApplyChanges()` method.

The following event-handler tests for BOGUS Customer's existence, updates the `CompanyName` and `ContactName` properties, and verifies the change:

#### C# 3.0

```
// Edit BOGUS Customer
private void btnEditCust_Click(object sender, EventArgs e)
{
    // Generate the proxy, delete the BOGUS Customer and SaveChanges
    NorthwindEntities proxy = new NorthwindEntities(new
        Uri("http://localhost.:54019/Northwind.svc", UriKind.Absolute));

    // Test for presence of BOGUS Customer with a query-string URI
    string query = "Customers?$filter=CustomerID eq 'BOGUS'";
    IEnumerable<Customer> bogus = proxy.Execute<Customer>(
        new Uri(query, UriKind.Relative));

    // Test for presence before editing
    foreach (var b in bogus)
    {
        b.CompanyName = "Bogus Software Corp.";
        b.ContactName = "Joseph Bogus, Sr.";
        proxy.UpdateObject(b);
        proxy.SaveChanges(SaveChangesOptions.ReplaceOnUpdate);

        // Verify edits
        IEnumerable<Customer> bogus2 = proxy.Execute<Customer>(
            new Uri(query, UriKind.Relative));
        foreach (var f in bogus2)
        {
```

## Chapter 15: Using the Entity Framework as a Data Source

---

```
        txtResult.Text += "\r\nEdited Customer data with: \r\n" +
            "CustomerID = " + f.CustomerID + "\r\n" +
            "CompanyName = " + f.CompanyName + "\r\n" +
            "ContactName = " + f.ContactName + "\r\n";
        return;
    }
}
txtResult.Text = "BOGUS Customer isn't present.";
}
```

### VB 9.0

```
' Edit BOGUS Customer
Private Sub btnEditCust_Click(ByVal sender As Object, ByVal e As EventArgs)
    ' Generate the proxy, delete the BOGUS Customer and SaveChanges
    Dim proxy As New NorthwindEntities(New Uri("http://localhost.:54019/Northwind.svc", UriKind.Absolute))

    ' Test for presence of BOGUS Customer with a query-string URI
    Dim query As String = "Customers?$filter=CustomerID eq 'BOGUS'"
    Dim bogus As IEnumerable(Of Customer) = _
        proxy.Execute(Of Customer)(New Uri(query, UriKind.Relative))

    ' Test for presence before editing
    For Each b In bogus
        b.CompanyName = "Bogus Software Corp."
        b.ContactName = "Joseph Bogus, Sr."
        proxy.UpdateObject(b)
        proxy.SaveChanges(SaveChangesOptions.ReplaceOnUpdate)

    ' Verify edits
    Dim bogus2 As IEnumerable(Of Customer) = _
        proxy.Execute(Of Customer)(New Uri(query, UriKind.Relative))
    For Each f In bogus2
        txtResult.Text += "Edited Customer data with: " & vbCrLf & _
            "CustomerID = " & f.CustomerID & vbCrLf & _
            "CompanyName = " & f.CompanyName & vbCrLf & _
            "ContactName = " & f.ContactName & Constants.vbCrLf
    Next f
    Next b
    txtResult.Text = "BOGUS Customer isn't present."
End Sub
```

Specifying the `SaveChangesOptions.ReplaceOnUpdate` enumeration as `proxy.SaveChanges()`'s optional argument causes only the changed data to be sent on the wire. Omitting the argument or specifying `SaveChangesOptions.None` sends values for all fields.

## Part V: Implementing the ADO.NET Entity Framework

---

### Deleting Entities

The following event-handler tests for BOGUS Customer's existence, deletes the instance and its persistent record, and verifies the change:

#### C# 3.0

```
// Delete BOGUS Customer
private void btnDeleteCust_Click(object sender, EventArgs e)
{
    // Generate the proxy, delete the BOGUS Customer and SaveChanges
    NorthwindEntities proxy = new NorthwindEntities(new
        Uri("http://localhost.:54019/Northwind.svc", UriKind.Absolute));
    // Test for presence of BOGUS Customer with a query-string URI

    string query = "Customers?$filter=CustomerID eq 'BOGUS'";
    IEnumerable<Customer> bogus = proxy.Execute<Customer>(
        new Uri(query, UriKind.Relative));

    // Test for presence before deleting
    foreach (var b in bogus)
    {
        proxy.DeleteObject(b);
        Dim dsrDel As DataServiceResponse = Nothing
        dsrDel = proxy.SaveChanges()

        // Verify deletion
        IEnumerable<Customer> bogus2 = proxy.Execute<Customer>(
            new Uri(query, UriKind.Relative));
        foreach (var d in bogus2)
        {
            txtResult.Text = "BOGUS Customer deletion failed.";
            return;
        }
        txtResult.Text += "BOGUS Customer deleted.\r\n";
        return;
    }
}
```

#### VB 9.0

```
' Delete BOGUS Customer
Private Sub btnDeleteCust_Click(ByVal sender As Object, ByVal e As EventArgs)
    ' Generate the proxy, delete the BOGUS Customer and SaveChanges
    Dim proxy As New NorthwindEntities(New Uri("http://localhost.:54019/Northwind
.svc", UriKind.Absolute))
    ' Test for presence of BOGUS Customer with a query-string URI

    Dim query As String = "Customers?$filter=CustomerID eq 'BOGUS'"
    Dim bogus As IEnumerable(Of Customer) = _
        proxy.Execute(Of Customer)(New Uri(query, UriKind.Relative))

    ' Test for presence before deleting
    For Each b In bogus
        proxy.DeleteObject(b)
```

## Chapter 15: Using the Entity Framework as a Data Source

---

```
Dim dsrDel As DataServiceResponse = Nothing
dsrDel = proxy.SaveChanges()

' Verify deletion
Dim bogus2 As IEnumerable(Of Customer) =
    proxy.Execute(Of Customer)(New Uri(query, UriKind.Relative))
For Each d In bogus2
    txtResult.Text = "BOGUS Customer deletion failed."
    Return
Next d
txtResult.Text &= "BOGUS Customer deleted."
Return
Next b
End Sub
```

### Managing Optimistic Concurrency Conflicts

Astoria uses Entity Tags (ETags), which the HTTP 1.1 specification's section 3.11 states "are used for comparing two or more entities from the same requested resource," to provide database values to test before applying an update. ETags appear in `<entry>` elements of multiple-entry `<feed>` documents as `m:etag` attributes with a quoted, comma-separated string value for each `EntityObject` field whose `ConcurrencyMode` property value has been changed from `None` to `Fixed` in the EDM Designer. For example, following is the ETag for the first Northwind Customer entity with its `Address`, `City`, `CompanyName`, `ContactName`, and `ContactTitle` property values specified for concurrency testing:

```
<entry m:etag="W/ ''Obere%20Str.%2057', 'Berlin', 'Alfreds%20Futterkiste',_
      'Maria%20Anders', 'Sales%20Representative'''>
<id>http://localhost:54019/Northwind.svc/Customers('ALFKI')</id>
<title type="text" />
...
</entry>
```

A similar ETag: `W/ ''Obere%20Str.%2057', 'Berlin', 'Alfreds%20Futterkiste', 'Maria%20Anders', 'Sales%20Representative''` item appears in the HTTP Entity Response Headers of single documents that have a single `<entry>`. The Astoria client compares the ETag database values with original values stored by the data source's `ObjectContext` and throws an error if they don't match.

*The `W/` prefix specifies a weak entity tag, which the specification says "MAY be shared by two entities of a resource only if the entities are equivalent and could be substituted for each other with no significant change in semantics."*

### Batching Changes

The preceding examples assume that each `AddToEntityType()`, `UpdateObject()` or `DeleteObject()` invocation has an immediately following `SaveChanges()` method call. You can accumulate changes on the local `DataServiceContext` and then invoke `DataServiceContext.SaveChanges(SaveChangesOptions.Batch)`, which creates a single `ChangeSet` and one HTTP request for all updates. The data source wraps the `ChangeSet` in a transaction; if all changes succeed, the transaction commits; otherwise, it rolls back.

## Consuming ADO.NET Data Services with the AJAX Client Library

ASP.NET 3.5 SP1 now includes the AJAX framework, which remains a separate download for ASP.NET 2.0. VS 2008 SP1 includes greatly improved JavaScript support with IntelliSense and a new AJAX Web Form template, as well as templates for AJAX Client Behavior, AJAX Client Control, the AJAX Client Library, the AJAX Master Page, and AJAX-enabled WCF Service components. Adding an AJAX Web form from the template to your Web application or Web site greatly simplifies the process of creating JavaScript-ready pages that use the ASP.NET ScriptManager control.

However, VS 2008 SP1 doesn't include the ADO.NET Data Services AJAX Client Library, which you must download as `AJAXClientForADONETDataServices.zip` from <http://www.codeplex.com/aspnet/Release/ProjectReleases.aspx?ReleaseId=13357>. The `.zip` file contains `DataService.debug.js` and `DataService.js` libraries, one of which you must copy to your project's folder and include in the project. `DataService.debug.js` is intended for use in development, and `DataService.js` is the runtime (release) version.

### Becoming Familiar with JavaScript Object Notation

The default data interchange format for JavaScript isn't XML; it's JavaScript Object Notation (JSON). According to <http://www.json.org>, JSON is a language-independent text format that consists of a collection of comma-separated name-value pairs that represent an object, which is wrapped with French braces (`{ }`). Values can be strings enclosed by double-quotes ("`chars`"), numbers, `true`, `false`, or `null`. Strings use reverse solidus (backslash) escapes. Arrays of objects, which are wrapped by square brackets (`[ ]`), contain multiple object instance representations. The advantage of JSON over XML is that standards-compliant JavaScript supports the generation of objects from data simply by applying the `eval()` operator to JSON strings.

Here's a simple ASP.NET AJAX `WebRequest` to return the response in JSON format for the first two Northwind U.S. Customer entities and enable viewing the wire data in Fiddler2:

```
function getCustomers() {
    var wr = new Sys.Net.WebRequest();
    wr.set_url('http://localhost:54019/Northwind.svc/ & _'
        "Customers?$filter=Country eq 'USA'&$top=2");
    wr.get_headers()["accept"] = "application/json";
    wr.add_completed(onComplete);
    wr.invoke();
}
```

*As mentioned earlier in the chapter, localhost's emphasized period suffix enables Fiddler2 to display data for calls made to 127.0.0.1.*

## Chapter 15: Using the Entity Framework as a Data Source

---

Following is the JSON response to the preceding function call with important elements highlighted:

### JSON

```
HTTP/1.1 200 OK
Server: ASP.NET Development Server/9.0.0.0
Date: Mon, 15 Sep 2008 17:06:06 GMT
X-AspNet-Version: 2.0.50727
DataServiceVersion: 1.0;
Cache-Control: no-cache
Content-Type: application/json; charset=utf-8
Content-Length: 1089
Connection: Close

{ "d" : [{"__metadata": {
"uri": "http://localhost.:54019/Northwind.svc/Customers('GREAL')",
"type": "NorthwindModel.Customer"}, "Address": "2732 Baker Blvd.", "City": "Eugene", "CompanyName": "Great Lakes Food Market", "ContactName": "Howard Snyder", "ContactTitle": "Marketing Manager", "Country": "USA", "CustomerID": "GREAL", "Fax": null, "Phone": "(503) 555-7555", "PostalCode": "97403", "Region": "OR", "Orders": {"__deferred": { "uri": "http://localhost.:54019/Northwind.svc/Customers('GREAL')/Orders" }}}, {"__metadata": {
"uri": "http://localhost.:54019/Northwind.svc/Customers('HUNGC')",
"type": "NorthwindModel.Customer"}, "Address": "City Center Plaza 516 Main St.", "City": "Elgin", "CompanyName": "Hungry Coyote Import Store", "ContactName": "Yoshi Latimer", "ContactTitle": "Sales Representative", "Country": "USA", "CustomerID": "HUNGC", "Fax": "(503) 555-2376", "Phone": "(503) 555-6874", "PostalCode": "97827", "Region": "OR", "Orders": {"__deferred": { "uri": "http://localhost.:54019/Northwind.svc/Customers('HUNGC')/Orders" }}}]
```

The Content-Type: application/json; charset=utf-8 indicates to the data service that the client expects a JSON-serialized response. d is an arbitrary object name. \_\_deferred and its uri replace the associated Orders EntityCollection if you don't add &\$expand=Orders to the query string URI; otherwise, the JSON content will include nested object representations.

## Creating an AJAX Test Client for Astoria

Adding an AJAX Web page to an existing Astoria Web service makes it simpler to use the ASP.NET Development Server than creating an independent client because the test project is self-contained. In this case, you can start the project by making a copy of the NwindDataServicesCS or VB project and adding a new AJAX Web Page named Default or the like to the project.

*This section's sample project is NwindDataServicesAjaxClientCS.sln and . . . VB.sln in the \WROX\ADONET\Chapter15 folder.*

## Part V: Implementing the ADO.NET Entity Framework

---

Download `AJAXClientForADONETDataServices.zip` as described in the earlier “Consuming ADO.NET Data Services with the AJAX Client Library” section, extract `DataService.debug.js` to the project folder, and include it in your project. The `DataService.[debug].js` library registers the `Sys.Data` namespace, and several classes, including the top-level `Sys.Data.DataService` class with `query()`, `insert()`, `update()`, and `remove()` methods.

The AJAX Web Page template adds a `<div>` element with an ASP.NET ScriptManager control. Add the following emphasized HTML code to install the AJAX Client Library, add a title, provide a button to call the `GetCustomers()` function you'll write, and include a named `<div>` to hold a table generated from the Northwind Customer EntitySet:

### HTML

```
<div>
    <asp:ScriptManager ID="ScriptManager1" runat="server">
        <Scripts>
            <asp:ScriptReference Path="~/DataService.debug.js" />
        </Scripts>
    </asp:ScriptManager>
    <h3 style="font-family: Calibri; font-size: 12pt; ">
        ADO.NET Data Services AJAX Client Demonstration
    </h3>
    <input type="button" id="btnGetData"
        value="Click to Generate Brazilian Customers Table"
        onclick="GetCustomers()" style="width: 290px; margin-bottom:10px;" />
    <div id="CustomersTable" style="font-family:Calibri;
        font-size:10pt; margin-bottom:10px;" />
</div>
```

Add the following JavaScript code after the built-in `pageLoad()` function to create a new `Sys.Data`.`DataService` proxy with same TCP port as the data source you copied and issue an asynchronous query-string HTTP GET request for the `Customer` entities in Brazil:

### JavaScript

```
function GetCustomers() {
    // Create a service proxy and issue an asynchronous query
    var service = new Sys.Data.DataService("/Northwind.svc");
    service.query("Customers?$filter=Country eq 'Brazil'", cbGotCustomers, cbFail);
}
```

*If you want to monitor HTTP traffic with Fiddler2, substitute `http://localhost.:55555/Northwind.svc` for `/Northwind.svc` as the service constructor's argument, changing 55555 to your service's fixed TCP port number.*

The `cbGotCustomers` and `cbFail` arguments are the names of the callback functions to handle a successful or failed query request, respectively. The following code in the `cbGotCustomers` callback

## Chapter 15: Using the Entity Framework as a Data Source

---

handler transforms the JSON string returned by the query into an HTML table with column names in headers and customer data in rows:

### JavaScript

```
function cbGotCustomers(result) {
    var sbTable = new Sys.StringBuilder("<table>");
    hasNewCustomer = false;
    var firstRow = true;
    for (i = 0; i < result.length; i++) {
        var row = result[i];

        if (firstRow) {
            // Display the header row
            sbTable.append("<tr>");
            for (key in row) {
                if (key != "__metadata" && key != "Orders") {
                    sbTable.append("<th>");
                    sbTable.append(key);
                    sbTable.append("</th>");
                }
            }
            sbTable.append("<tr>");
            firstRow = false;

            // Display the data
            sbTable.append("<tr>");
            for (key in row) {
                if (key != "__metadata" && row[key] != "[object Object]") {
                    sbTable.append("<td>");
                    sbTable.append(row[key]);
                    sbTable.append("</td>");
                    if (row[key] == "TARGI")
                        hasNewCustomer = true;
                }
            }
            sbTable.append("</tr>");
        }
        sbTable.append("</table>");
        $get("CustomersTable").innerHTML = sbTable.toString();
    }
}
```

`hasNewCustomer` is a page-scoped `bool` variable that update functions — the subject of the next section — use to add, edit and remove a TARGI customer. Following is a typical callback handler for a failed request:

### JavaScript

```
function cbFail(result) {
    alert("Request to the Northwind Data Service failed.");
}
```

## Part V: Implementing the ADO.NET Entity Framework

Figure 15-11 shows the table returned by the preceding code after adding a new Brazilian customer, TARGI, in Itajaí, Santa Catarina (SC).

Address	City	CompanyName	ContactName	ContactTitle	Country	CustomerID	Fax	Phone	PostalCode	Region
Av. dos Lusiadas, 23	Sao Paulo	Comércio Mineiro	Pedro Afonso	Sales Associate	Brazil	COMMI		(11) 555-7647	05132-043	SP
Rua Orós, 92	Sao Paulo	Família Arquibaldo	Aria Cruz	Marketing Assistant	Brazil	FAMIA		(11) 555-9857	05442-030	SP
Av. Brasil, 447	Campinas	Gourmet Lanchonetes	André Fonseca	Sales Associate	Brazil	GOURI		(11) 555-9482	04876-786	SP
Rua do Paço, 67	Rio de Janeiro	Hanari Carnes	Mario Pontes	Accounting Manager	Brazil	HANAR	(21) 555-8765	(21) 555-0091	05454-876	RJ
Rua da Panificadora, 12	Rio de Janeiro	Que Delícia	Bernardo Batista	Accounting Manager	Brazil	QUEDE	(21) 555-4545	(21) 555-4252	02389-673	RJ
Alameda dos Canários, 891	Sao Paulo	Queen Cozinha	Lúcia Carvalho	Marketing Assistant	Brazil	QUEEN		(11) 555-1189	05487-020	SP
Av. Copacabana, 267	Rio de Janeiro	Ricardo Adocicados	Janete Limeira	Assistant Sales Agent	Brazil	RICAR		(21) 555-3412	02389 890	RJ
Rua Dr Pedro Ferreira 155	Itajaí	Target Importação e Exportação	Max Heiden	Owner	Brazil	TARGI	47 3045-7003	47 3045-7002	88301-030	SC
Av. Inês de Castro, 414	Sao Paulo	Tradição Hipermercados	Anabela Domingues	Sales Representative	Brazil	TRADH	(11) 555-2168	(11) 555-2167	05634-030	SP
Rua do Mercado, 12	Resende	Wellington Importadora	Paula Parente	Sales Manager	Brazil	WEILL		(14) 555-8122	08737-363	SP

Figure 15-11

*Itajaí is on the Atlantic coast, about halfway between São Paulo and Porto Allegre.*

## Adding, Editing, and Removing Entities

Updating entities with JSON follows the same programming pattern used for AtomPub. Adding a new customer requires creating a newCustomer object and invoking the `DataService.insert()` method, as shown here:

### JavaScript

```
// Add a new TARGI customer
function AddCustomer() {
    if (!hasNewCustomer)
    {
        alert("New customer has already been added.\r\nClick OK to continue");
        return
    }
}
```

## Chapter 15: Using the Entity Framework as a Data Source

---

```
    }
    var service = new Sys.DataDataService("/Northwind.svc");
    var newCustomer = {
        CustomerID : "TARGI",
        CompanyName : "Target Importação e Exportação",
        ContactName : "Max Heiden",
        ContactTitle : "Owner",
        Address : "Rua Dr Pedro Ferreira 155",
        City : "Itajaí",
        Region : "SC",
        PostalCode : "88301-030",
        Country : "Brazil",
        Phone: "47 3045-7002",
        Fax: "47 3045-7003"
    };
    try {
        service.insert(newCustomer, "/Customers", GetCustomers, cbFail);
    }
    catch (ex) {var msg = "An error occurred adding the new customer.\n\n";
        + "Error description: " + ex.description + "\n\nClick OK to continue.";
        alert(msg);
    }
}
```

Editing an entity requires retrieving it from the `EntitySet`, replacing property values, and invoking the `DataService.update()` method with the edited entity, as illustrated by:

### JavaScript

```
// Edit the TARGI customer's ContactName and ContactTitle
function EditCustomer()
{
    if (hasNewCustomer) {
        var service = new Sys.DataDataService("Northwind.svc");
        service.query("/Customers('TARGI')", cbRetrieved, cbFail);
    }
    else
        alert("New customer has not been added.\r\nClick OK to continue");
}

function cbRetrieved(result) {
    var editCustomer = result;
    editCustomer.ContactName = "Joao Texiera";
    editCustomer.ContactTitle = "General Manager";
    var service = new Sys.DataDataService("Northwind.svc");
    service.update(editCustomer, "\Customers", GetCustomers, cbFail);
}
```

## Part V: Implementing the ADO.NET Entity Framework

---

Deleting an entity requires invoking the `DataService.remove()` method with the primary-key value of the item to be deleted as the second argument, as in:

### JavaScript

```
// Delete the TARGI customer
function DeleteCustomer() {
    if (hasNewCustomer) {
        var service = new Sys.Data.DataService("Northwind.svc");
        service.remove(null, "/Customers('TARGI')", GetCustomers, cbFail);
    }
    else {
        alert("New customer has not been added.\r\nClick OK to continue");
    }
}
```

Each successful operation calls the `GetCustomers()` callback handler to refresh the table.

## Summary

The Entity Framework is the Microsoft Data Programmability group's first concrete implementation of the Entity Data Model, but there are many other data-related implementations in the works, starting with Reporting Services and continuing with Data Synchronization Services and, possibly, a future version of SQL Server Data Services. To encourage use of EF as a data source, the DP group made EF the only out-of-the-box data source for the ADO.NET Data Services Framework (Astoria) that enables updating entities. Because of minor incompatibilities with EF v1, LINQ to SQL is the preferred data source for ASP.NET Dynamic Data.

Astoria is a Representational State Transfer (REST) front end for data sources that support `IQueryable<T>` and, for read/write operations, `IUpdatable<T>` interfaces. REST identifies data items (resources) by a unique Uniform Resource Indicator (URI) and accesses them by HTTP without requiring an additional messaging layer, such as SOAP. Astoria's basic syntax for accessing `EntitySets` or `EntityObjects` is:

```
http://host/<service>/<EntitySet>[(<Key>) /<NavigationProperty>[(<Key>)/...]]]
```

For example, the following URL returns the `Customers` `EntitySet` from the `Northwind.svc` Windows Communication Framework (WCF) Service:

```
http://localhost/Northwind.svc/Customers/
```

And this URL returns the `Customer` entity with a primary-key value of `ALFKI`, together with its associated `Orders` `EntityCollection`:

```
http://localhost/Northwind.svc/Customers('ALFKI')/Orders
```

URIs can include query-string options, such as `expand`, `filter`, `orderby`, `skip`, `top`, `metadata`, and `value` to customize the returned entities. Astoria supports HTTP's GET method for queries, POST for inserts, PUT for updates, and DELETE for deletions.

## Chapter 15: Using the Entity Framework as a Data Source

---

Astoria's serialization format for entities is the Atom Publication Protocol (AtomPub or APP) specification, which is the subject of Internet Engineering Taskforce (IETF) Request for Comments (RFC) 5023. .NET 3.5 SP1's WCF component supports SOAP, AtomPub, and JavaScript Object Notation (JSON) wire formats. AtomPub is extensible, so the Astoria team has supplemented the standard with customized representations of entities associated in one:many and many:one relationships, as well as HTTP ETags for concurrency conflict management.

The lightweight JSON protocol is better suited for ASP.NET Asynchronous JavaScript and XML (AJAX) Web projects because developers can parse it with JavaScript to generate HTML objects, such as `tables` and `select` lists or apply the `eval` operator to regenerate objects.

Sample projects demonstrate basic Astoria Web services with C# and VB Windows form clients that use the .NET Framework 3.5 Client Library process AtomPub messages and AJAX Client Library for Web projects that create and edit a table populated by a JSON stream. A Silverlight 2 library also is available for Astoria. The client libraries supplement query-string URI requests with LINQ to REST queries, which support a subset of the LINQ Standard Query Operators.



# Index

## Numbers

1:Many associations, 290–291

## A

**abstract base types**, 425–427

**abstraction, EDM**

adding, updating, and deleting model objects, 367–368

creating XML mapping files and object layer class files, 365–367

editing `EntityType` and `AssociationSet` properties, 368–369

EF (Entity Framework) and, 363–364

`EntityQueries` in Entity SQL, 376–381

entity-relationship model, 359–362

examining `ConceptualModels` group, 371–373

mapping physical layer to conceptual layer, 365

Object Services. See Object Services overview, 357–359

Persistence Ignorance controversy and, 390–391

scanning `StorageModels` group, 370–371 summary, 391–392

tracing `Mappings` group, 374–375

**abstraction, leaky**, 210

**Active Directory**. See **LINQ to Active Directory**

**active entity instances**, 540–543

**Active Record pattern**, 391

**adding collection members**, 237

**adding entities**, 509, 602–604

**adding objects, LINQ to SQ**, 215–219

**addition operations**

for updates, 549–550

validating, 510–513

**ADO.NET**

Data Services. See Data Services

Data Source. See data sources

Entity Framework. See EF (Entity Framework) query string options. See query string options

**advanced queries**

aggregate operators. See aggregate operators

`Contains()` SQO. See `Contains()` SQO

`GroupBy`, with aggregate queries. See `Group By`, with aggregate queries

mocking collections for testing projects. See mocking collections

overview, 155

query expression trees, 172–176

summary, 187–188

**AdventureWorksMetadata VB project**, 428–429

**aggregate keyword, with VB**, 161–162

**aggregate operators**

`Aggregate`, 152–153, 159

`Count` and `LongCount`, 150

`Let` variable, 158–159

method call syntax, 157–158

`Min`, `Max`, `Sum`, and `Average`, 151

overview, 149, 156

query output formatting, 159–160

**aggregate values, for WHERE clause constraints**, 454–455

`Aggregate...` `Into`

in advanced query expressions, 159

in grouping with associated child objects, 161–162

**aggregating business documents**. See **business documents, grouping/aggregating**

**aggregating child entity values**, 158

**AJAX Client Library**

adding, editing and removing entities, 602–604

becoming familiar with JSON, 598–599

creating an AJAX test client, 599–602

overview, 598

**alias prefixes, in Entity SQL and Transact-SQL**, 440–441

**All**, as a quantifier operator, 148–149

**Amazon, LINQ to Amazon**, 7

**Ancestors, XML extension method**, 271

**AncestorsAndSelf, XML extension method, 271**

**ANDD (ASP.NET Dynamic Data)**  
EF as a data source for, 568  
overview, 4

**ANDS (ADO.NET Data Services). See Data Services**

**anonymous delegates, 75**

**anonymous methods**  
C# 2.0, 75–76  
C# 3.0, 77  
generic predicates and, 75–77  
VB 8.0, 76, 77

**anonymous structs, 269**

**anonymous types**  
C# 3.0, 71  
DataSets, 263–266  
overview, 60  
VB 9.0, 71–72

**Any operator, 148**

**ANYELEMENT operator, 452**

**AppendOnly, merge option, 482**

**arithmetic operators, 333–334, 578**

**array initializers, 70**

**As DataView(), 257–260**

**AsEnumerable operator, 133**

**ASP.NET databinding**  
adding a LinqDataSource to a page, 226–228  
eager loading EntityRef values for, 231–233  
overview, 226  
substituting EntityRef for ForeignKey values, 228–230

**ASP.NET Dynamic Data (ANDD)**  
EF as a data source for, 568  
overview, 4

**ASP.NET EntityDataSource server control. See EntityDataSource server control**

**ASP.NET Web application page, 555–557**

**AsQueryable operator, 133–135**

**assemblies, installing LINQ to SharePoint, 344–345**

**associated entities**  
dependent, 508  
EntityCollections, 489–491  
EntityObject, 507–508  
EntitySet, 507–508  
lazy loading, 338–340

returning, 587–589  
selecting instances of, 540–543

**associated entities, deferred-loaded**  
enabling, 483–489  
entity class serialization with, 478  
with the Load() method, 484–485

**associated entities, eager-loaded**  
enabling, 483–489  
entity class serialization with, 478–479  
with Include() operators, 485–489

**associated objects**  
deferred loading with Attach() method, 539–540  
deferred loading with Load() method, 537–538  
eager loading with Include() method, 538–539  
equi-joins of, 110–111  
grouping, 161–162  
SelectMany implementation, 108–109

**Association subgroups**  
in ConceptualModels (CSDL content), 414  
overview, 409  
population of associations with stored procedures, 410–411

**associations**  
1:Many and Many:1, 290–291  
C# 3.0, 24  
keys, 239–240  
with LINQ join...on expressions, 27–29  
mapping, 407–408  
navigating entity set, 37–38  
navigating to emulate joins, 24–27  
populating with stored procedures, 410–411  
sets, 545–546  
unbound ComboBoxes to specify, 543–546  
VB 9.0, 25–26

**AssociationSet**  
elements or subgroups, 398–399  
Mapping subgroup, 417–418  
properties, editing, 368–369  
selecting and adding new, 551–553

**Astoria**  
creating AJAX test client for, 599–602  
query string options, 576–580

**Attach() method**  
deferred loading with, 539–540  
deferred loading with the Attach() method, 539–540

**attributes**  
attribute-centric mapping, 20, 210

EDM terminology definition, 361  
 implicit v. explicit typing of attribute content, 276–278  
 in XML extension method, 271  
**AutoGenerateOrderByClause** **property**, EntityDataSource, **557**  
**AutoGenerateOrderWhereClause** **property**, EntityDataSource, **557**  
**autoincrementing primary key**, **237**  
**AutoPage** **property**, EntityDataSource, **557**  
**AutoSort** **property**, EntityDataSource, **557**  
**Average operator**, **151**  
**axis properties**, VB 9.0, **275–276**

## B

**bad request errors**, **338–340**

**base classes, querying**

disambiguate derived object types with base types, 425–427  
 incorrect results from `is`, `Is`, `TypeOf` and `OfType` operators, 423–424  
 overview, 421–423  
 type discrimination in Entity SQL queries, 424–425

**batching**

changes in, 597  
 queries, 589–591

**binding data**

with `AsDataView()`, 257–260  
 databound web controls, 228–230  
 LINQ, 12–15  
 overview, 533–534  
 summary, 565–566  
 with `ToList()` operator, 12–15  
 using `EntityDataSource`. See `EntityDataSource` server control  
 Windows form controls to entities. See Windows form controls, binding to entities

**binding data, ASP.NET**

adding a `LinqDataSource` to a page, 226–228  
 eager loading `EntityRef` values for, 231–233  
 overview, 226  
 substituting `EntityRef` for `ForeignKey` values, 228–230

**browser, creating/consuming Web service in**, **571–572**

**business documents, grouping/aggregating**

aggregating `Order_Details` and `OrdersSubtotals`, 292–294

`GroupJoin` to produce hierarchical documents, 289  
 overview, 288  
 taking advantage of `1:Many` and `Many:1` associations, 290–291

## C

**C#**

functional construction with C# 3.0, 280–284  
 PLINQ, 324–325  
 XML namespaces in C# 3.0. See XML namespaces, in C# 3.0

**C# extensions**

anonymous methods and generic predicates, 75–77  
 anonymous types, 71–72  
 array initializers with object initializers, 70  
 collection initializers, 70–71  
 expression trees and compiled queries, 83–88  
 extension methods, 72–75  
 implicitly typed local variables, 67–68  
`IQueryable<T>` implementations, 88–89  
 lambda expressions, 77–80  
 object initializers, 68–69  
 overview, 66–67  
 standard query operators, 80–83

**CAML (Collaborative Application Markup Language)**, **348–349**

**canonical functions**, **439–440**

**Canonical Query Tree (CQT)**

in ADO.NET Data Services, 569  
 definition, 27, 363  
 in EF architecture, 364

**cascading deletions, in SQL Server**, **238–239**

**Cast operator**, **135–136**

**changes**

persisting, 506  
 tracking, 20

**child element groups**, **308–312**

**child entity values**, **158**

**classes, base**, **421–423**

**classes, C#, 17–19, 98–100**

**classes, creating mock object**, **177–182**

**classes, derived**

creating, 419–421  
 disambiguate derived object types with base types, 425–427  
 querying, 421–423

## classes, entity

C# 3.0 and VB 9.0, 470  
EntityName partial classes, 472–477  
generated, 470  
ModelNameEntities partial classes, 471–472  
serialization, 477–479

## classes, generated

C#, 17–19  
examining, 206–210  
VB 9.0, 17, 19–20

## classes, LINQ to SQL, 16–20, 206–210

## classes, partial entity

EntityName, 472–477  
generating with SqlMetal.exe, 200–203  
ModelNameEntities, 471–472

## classes, VB 9.0, 17, 19–20, 100–101

## clause constraints, 454–455

## client layer, ADO.NET Data Service, 569–570

## Client Library. See .NET 3.5 Client Library

## client projects. See Northwind.svc WCF Service

### example

## client views

EF components, 363–364  
executing Entity SQL queries as, 376–381  
working with, 375

## clients, entity

queries against. See Entity SQL, queries against EntityClient  
queries as client views. See Entity SQL, queries as client views  
working with client views, 375  
writing EntityQueries in Entity SQL, 376

## CLR (Common Language Runtime)

integrating XML into, 267–269  
strongly typed DataSets stored as, 212

## code

implementing optimistic concurrency, 517–520  
sample classes for C# and VB, 98–100  
writing LINQ to SharePoint, 346–348

## Collaborative Application Markup Language (CAML), 348–349

## Collation, SQL Server facet name, 401

## collections

adding members, 237  
ANYELEMENT and FLATTEN operators, 452  
DeleteOnNull attribute, 239–240  
deleting dependent records, 240–242  
deleting members, 238  
initializers, 60, 70–71  
mocking. See mocking collections

navigating, 573–576

SQL Server cascading deletions, 238–239

## column, discriminator, 419–421

## ComboBoxes

event handlers for ComboBoxColumns, 543–544  
setting composite primary-key members. See composite primary keys  
specifying associations, 543–546

## command-line options, with SqlMetal.exe, 201–202

## CommandText property,

EntityDataSource, 557

## Common Language Runtime (CLR)

integrating XML into, 267–269  
strongly typed DataSets stored as, 212

## comparison operators, 578

## compiled queries

code examples, 87–88  
LINQ to Entities, 497–499  
overview, 87

## complex types

overview, 528  
reusing a ComplexType definition, 530–531  
working with, 528–531

## complex types, modeling

modifying the CSDL section, 529  
modifying the MSL section, 530  
overview, 529  
reusing a ComplexType definition, 530–531

## composite primary keys

detecting attempts to change values, 546–549

enforcing object deletion and addition, 549–550

overview, 546

selecting/adding EntityObject and AssociationSet, 551–553

## compound OR operators, 169–171

## compound Where expressions, 102–104

## Concat operator, 120–121

## conceptual database schema, 50–52

## Conceptual Schema Definition Language (CSDL), 529. See also ConceptualModels (CSDL)

## conceptual schema, mapping physical to

adding, updating, and deleting objects, 367–368  
creating the XML mapping files and object layer class file, 365–367

- editing `EntityType` and `AssociationSet` properties, 368–369  
 overview, 365
- ConceptualModels (CSDL)**  
 Association subgroup, 414  
 EntityContainer subgroup, 411–412  
`EntityType` subgroup, 413  
 group, 371–373  
 overview, 411
- concurrency**  
 detecting and resolving conflicts, 219–222  
 managing optimistic conflicts, 515–520, 597  
 managing pessimistic, 515  
 managing with original values, 525–527  
 managing with timestamp property, 527–528
- Connection page, for eSqlBlast, 434–435**
- ConnectionString property,**  
`EntityDataSource`, 557
- construction**  
`C# XDocuments`, 300–302  
`VB 9.0 in XML`, 284–288  
 XML documents, 44
- Contains() SQO**  
 emulating SQL `IN( )` function, 171–172  
 emulating SQL `Where` clauses, 169–171  
 overview, 169  
 as a quantifier operator, 149
- ContextTypeName property,**  
`EntityDataSource`, 557
- controls, data-enabled, 21–23**
- conversion operators**  
`AsEnumerable`, 133  
`AsQueryable`, 133–135  
`Cast`, 135–136  
`OfType`, 136–138
- Count operator, 150**
- CountValidISBNs, PLINQ, 324**
- CQT (Canonical Query Tree)**  
 in ADO.NET Data Services, 569  
 definition, 27, 363  
 in EF architecture, 364
- create, retrieve, update, and delete. See CRUD (create, retrieve, update, and delete)**
- CREATEFEF, entity reference, 446–449**
- CRUD (create, retrieve, update, and delete)**  
 creating initial Data Model, 521–522  
 deleting entity instances with stored procedures, 528  
`FunctionImports` to populate `EntitySets`, 522–524
- inserting a new entity instance, 524–525  
 Object Services and, 469  
 overview, 521, 524  
 summary, 532  
 updating an entity instance and managing concurrency with original values, 525–527  
 updating an entity instance and managing concurrency with timestamp property, 527–528
- CSDL (Conceptual Schema Definition Language), 529. See also ConceptualModels (CSDL)**
- Customer, 290–291**
- Cw , 61, 269**

## D

**DAL (data access layer), 569–570. See also LINQ toSQL, as a DAL**

### data

- `C# example of table data`, 31–32
- editing form, 233–236
- retrieval, 405–406
- shapes, 71
- storage. See data store
- updating table data, 30–32
- validating additions and updates, 510–513

VB example of table data, 32–34

### data abstraction, EDM

- adding, updating, and deleting model objects, 367–368
- creating XML mapping files and object layer class files, 365–367
- editing `EntityType` and `AssociationSet` properties, 368–369
- EF (Entity Framework) and, 363–364
- EntityQueries in Entity SQL, 376–381
- entity-relationship model, 359–362
- examining `ConceptualModels` group, 371–373
- mapping physical layer to conceptual layer, 365
- Object Services. See Object Services
- overview, 357–359
- Persistence Ignorance controversy and, 390–391
- scanning `StorageModels` group, 370–371
- summary, 391–392
- tracing `Mappings` group, 374–375

## data access

EDM (Entity Data Model). See EDM (Entity Data Model)  
EF (Entity Framework). See EF (Entity Framework)  
LINQ (Language Integrated Query). See LINQ (Language Integrated Query)  
overview, 3–5  
summary, 57–58

## data access layer (DAL), 569–570. See also LINQ toSQL, as a DAL

## Data Model, creating initial, 521–522

## Data Services

consuming with AJAX Client Library. See AJAX Client Library  
consuming with .NET 3.5 Client Library. See .NET 3.5 Client Library

## Data Services data source

creating Web service, 571–572  
invoking service operations, 580–581  
navigating collections and their members, 573–576  
overview, 568–570  
query string options, 576–580  
REST architecture and, 570

## Data Services Framework

data flow diagram, 327  
overview, 4  
query options, 333  
three-layered structure, 569–570

## data sources. See also EntityDataSource server control

binding controls to entity data sources, 21–23  
creating and binding to, 54–56  
Data Services. See Data Services data source  
EF (Entity Framework) as, 567–568, 604–605

## data store

persisting changes, 506, 553–554  
SQL Server Compact as, 461–463  
substituting SSCE for SQL Server, 462–463

## data table, 30–34

## database schema

\*.dbml and \*.xml files and, 202–206  
OR Designer generated, 34  
physical and conceptual, 49–52  
XML documents, 42–47

## database server traffic, reducing, 231–233

## databinding

with AsDataView (), 257–260  
LINQ, 12–15

overview, 533–534  
summary, 565–566  
with `ToList()` operator, 12–15  
using `EntityDataSource`. See `EntityDataSource` server control  
Windows form controls to entities. See Windows form controls, binding to entities

## databinding, ASP.NET

adding a `LinqDataSource` to a page, 226–228  
eager loading `EntityRef` values for, 231–233  
overview, 226  
substituting `EntityRef` for `ForeignKey` values, 228–230

## databound web controls, 228–230

`DataContext`  
C# 3.0, 23–25  
`DataSet`s compared with, 244–245  
emulating joins, 24–27  
emulating SQL functions, modifiers and operators, 30  
explicit joins, 27–29  
generating a SQL server database, 34  
instantiating, 210–211  
lazy or eager loading, 27  
`LinqDataSource` control, 34  
overview, 23–24  
updating table data, 30–34  
VB 9.0, 24, 25–26

## data-enabled controls, binding to entity data sources, 21–23

## DataRow, 262–263

## DataServiceQuery, 585–586

## DataServiceRequests, 589–591

## DataSet. See also LINQ to DataSet

C# 3.0, 35–37, 249–251  
`DataContext`s compared with, 244–245  
joining strongly typed, 35–36  
joining untyped, 36–37  
processing anonymous types, 263–266  
querying typed, 254–257  
querying untyped, 248–252  
T-SQL, 35  
VB 9.0, 36–37, 251–252

## DataMember, 427

## DataTable, copying LINQ query results to, 260–266

copying typed `DataRow`s, 262–263  
overview, 260–261

processing anonymous types from projections, 263–266

**DataTables, querying.** See **LINQ to DataSets**

**date query functions**, 335

**DateTime functions**, 579

**datetime Scalar Function**, 404–405

**DateTimeKind**, 401

\*.dbml files, 203–204

**Default, facet name**, 401

**DefaultContainer property**, EntityDataSource, 557

**DefaultIfEmpty operator**, 146

**deferred loading**

- of associated entities, 478, 483–489
- Attach() method, 539–540
- Load() method, 484–485, 537–538
- overview, 483

**delegates**

- anonymous, 75
- generic, 76–77

**DELETE stored procedure**

- assigned to entities, 528
- mapping to EntityTypes, 406–407
- substituting for dynamic SQL, 224–225

**DeleteOnNull, association key**, 239–240

**deleting collection members**

- DeleteOnNull attribute, 239–240
- dependent records, 240–242
- overview, 238
- SQL Server cascading deletions, 238–239

**deletion operations**

- in SQL Server, 238–239
- transacting, 508
- for updates, 549–550

**dependent associated entities**, 508

**dependent EntitySet**, 238–242

**dependent records**, 240–242

**dependent rows**, 240–242

**Deref, entity reference**, 446–449

**derived classes**

- creating, 419–421
- disambiguate derived object types with base types, 425–427
- querying, 421–423

**derived object types**, 425–427

**DescendantNodesAndSelf, in XML extension method**, 271

**DescendantNodes(T), in XML extension method**, 271

**Descendants, in XML extension method**, 271

**DescendantsAndSelf, in XML extension method**, 271

**Designer, EDM**

- EF roles, 363–364
- mapping physical to the conceptual schema, 50–52
- modifying storage to conceptual mapping with, 53–54
- update stored procedure association mapping in, 407–408

**Designer, O/R**

- editing \*.dbml files in, 203–204
- generating a SQL server database with, 34
- mapping tables to entity sets, 15–16, 197–200

**DetailsView, in EntityDataSource**, 564–565

**diagrams, entity-relationship**, 361–362

**Dim Query**, 6

**disambiguation, of derived object types**, 425–427

**discrimination, type**, 424–425

**discriminator column, TPH inheritance**, 419–421

**DISTINCT (eSQL)**, 491

**Distinct (QBM)**, 491

**Distinct operator, SQO**, 129

**documents**

- hierarchical, 289
- joining, 314–315
- joining to insert elements, 312–313
- multiple namespaces, 306–308

**documents, grouping/aggregating**

- aggregating Order\_Details and OrdersSubtotals, 292–294
- GroupJoin to produce hierarchical documents, 289
- overview, 288
- taking advantage of 1:Many and Many:1 associations, 290–291

**documents, XML**

- C# 3.0, 40–41, 42, 44–45
- querying, 40–44
- transforming or creating, 44–47, 306–308
- VB 9.0, 41–42, 43, 45–47

**domain objects**, 34

**domain-specific languages (DSLs)**, 63–64

# domain-specific LINQ implementations

---

## domain-specific LINQ implementations

application of, 190–191  
`IQueryable<T>` interface and, 88–89  
LINQ to Active Directory. See LINQ to Active Directory  
LINQ to DataSet. See LINQ to DataSets  
LINQ to SharePoint. See LINQ to SharePoint  
LINQ to SQL. See LINQ to SQL  
LINQ to XML. See LINQ to XML  
overview, 342

## dot-notation

`NAVIGATE` as a complex substitute for,  
445–446  
returning many:one navigation properties,  
442–443

## DropDownList, 563–564

## DSLs (domain-specific languages), 63–64

## Dynamic Data, ASP.NET, 4, 568

## dynamic SQL, 222–225

# E

## eager evaluation, in SQOs, 82

### eager loading

of associated entities, 478–479  
of child objects, 27  
`EntityRef` values to reduce server traffic,  
231–233  
`Include()` method, 485–489, 538–539  
overview, 483

## editing \*.dbml files, 203–204

## editing \*.xml file, 205–206

## editing entities, 236–237

### EDM (Entity Data Model)

creating default model, 52–53  
creating/binding to a data source, 54–56  
mapping physical to the conceptual schema,  
50–52  
materializing object context, 56–57  
modifying mapping fields to `EntityType`  
properties, 53–54  
overview, 48–50  
raising the level of data abstraction.  
See data abstraction, EDM

terminology and entity-relationship,  
360–361

### EDM Designer

EF roles, 363–364  
mapping physical to the conceptual schema,  
50–52

modifying storage to conceptual mapping  
with, 53–54  
update stored procedure association mapping  
in, 407–408

### EDM Wizard

creating default EDM with, 52–53  
creating the XML mapping files, 365–367

### EDMX file

`ConceptualModels` (CSDL content). See  
`ConceptualModels` (CSDL)  
`Mapping` (MSL content). See `Mapping` (MSL  
content)  
overview, 395–397  
`StorageModels` (SSDL content). See  
`StorageModels` (SSDL content)

### EF (Entity Framework)

architecture and components, comprehending,  
363–364  
consuming Data Services with AJAX Client  
Library. See AJAX Client Library  
consuming Data Services with .NET Client  
Library. See .NET 3.5 Client Library  
creating Data Services data source. See Data  
Services data source  
as a data source, 567–568, 604–605  
data store, 461–463  
Entity Framework futures, 467–468  
future of, 354–355  
history of, 353–354  
introduction to, 352–353  
LINQ to SQL compared with, 466–467  
mapping physical to the conceptual schema,  
50–52  
materializing an object context, 56–57  
`ModelName.edmx` file. See `ModelName.edmx`  
files, analyzing

### element operators

`DefaultIfEmpty`, 146  
`ElementAt` and `ElementAtOrDefault`, 146  
`First` and `FirstOrDefault`, 144–145  
`Last` and `LastOrDefault`, 145  
overview, 144  
`Single` and `SingleOrDefault`, 145–146

### elements

implicit v. explicit typing of element content,  
276–278  
joining documents to insert, 312–313  
in XML extension method, 271

**Empty operator, generation operator, 147**

**EnableDelete property,**

    EntityDataSource, **557**

**EnableInsert property,**

    EntityDataSource, **557**

**EnableUpdate property,**

    EntityDataSource, **558**

**EnableViewState property,**

    EntityDataSource, **558**

**entities. See also LINQ to Entities**

    adding, 592–594

    adding to collections, 237

    alias prefixes, 440–441

    binding controls to entity data sources,  
        21–23

    binding Windows forms controls to. See  
        Windows form controls, binding to entities

    C# 3.0, 37–38

    comparing performance, 514–515

    deleting, 505–508, 596–597

    deleting from collections, 238–242

    deleting with stored procedures, 528

    editing, 236–237

    EDM terminology definition, 360

    entity-relationship model, 360–362

    JSON for working with, 602–604

    left outer joins emulated with entity  
        associations, 168–169

    logging state of, 506

    in managing entity references, 446–449

    managing optimistic concurrency conflicts,  
        515–520

    mapping tables to, 15–16, 197–200

    ObjectStateManager and its children,  
        503–505

    optimizing the ObjectContext lifetime,  
        513–514

    persisting changes to the data store, 506

    refreshing stale, 509

    relationships and, 196

    selecting active top-level and associated  
        instances, 540–543

    stored procedure used to insert new instances,  
        524–525

    term usage, 244

    type constructors, 449–450

    updating, 505–506, 594–595

    updating instances of, 525–528

    validating data additions and updates, 510–513

    VB 9.0, 38

working with complex types, 528–531

**entities, associated**

    deferred-loaded, 478, 483–489

    dependent, 508

    eager-loaded, 478–479, 483–489

    EntityCollections, 489–491

    EntityObject, 507–508

    EntitySet, 507–508

    lazy loading, 338–340

    returning, 587–589

    selecting instances of, 540–543

**entity classes**

    C# 3.0 and VB 9.0, 470

    EntityName partial classes, 472–477

    generated, 470

    ModelNameEntities partial classes,  
        471–472

    serialization, 477–479

**entity clients**

    EF roles, 363–364

    queries against. See Entity SQL, queries against  
        EntityClient

    queries as client views. See Entity SQL, queries  
        as client views

    working with client views, 375

    writing EntityQueries in Entity SQL, 376

**Entity Data Model. See EDM (Entity Data Model)**

**Entity Data Model Wizard**

    creating default EDM, 52–53

    creating the initial EDM, 521–522

**Entity Data Platform, Microsoft, 393**

**Entity Framework. See EF (Entity Framework)**

**entity sets**

    adding FunctionImports to populate,  
        521–522

    adding new entities, 592–594

    associated entities and, 507–508

    batching changes, 597

    data retrieval stored procedures assigned to,  
        405–406

    deleting entities, 596–597

    deleting members from collections, 238–242

    EDM terminology definition, 360

    elements or subgroups, 398

    EntitySetMapping, 415–417

    EntitySetName, 558

    managing optimistic concurrency conflicts, 597

    mapping tables to, 197–200

    updating, 594–595

    updating associated, 507–508

# Entity SQL

---

## Entity SQL

EF roles, 363–364  
EntityQueries, 376  
ObjectQueries, 384–387, 480–483  
overview, 433–434  
query sample, 381  
query utility. See eSqlBlast  
SQL Server Compact used as an EF data store, 461–463  
statements for composing ObjectQueries, 387  
summary, 463–464  
Transact-SQL compared with. See Entity SQL/  
Transact-SQL comparison  
type discrimination in queries, 424–425  
using the eSqlBlast. See eSqlBlast

**Entity SQL, queries against** EntityClient  
executing parameterized queries, 460–461  
measuring the performance penalty of, 459  
overview, 455–456  
parsing the IExtendedDataRecord, 456–458

**Entity SQL, queries as client views**  
C# 3.0, 377–378  
Entity SQL from ecmdNwind.CommandText  
sample, 380  
overview, 376  
T-SQL, 381  
T-SQL from ToTraceString sample, 380–381  
VB 9.0, 378–380

**Entity SQL/Transact-SQL comparison**  
ANYELEMENT and FLATTEN operators, 452  
dot-notation syntax and, 442–443  
entity alias prefixes, 440–441  
explicit projections, 441  
IS OF, OFTYPE, and TREAT, 453–454  
JOINS, 445  
NAVIGATE, 445–446  
one:many navigation properties, 444–445  
ORDER BY clause for paging, 452–453  
overview, 439–440  
REF, DEREF, CREATEREF, ROW, and KEY  
references, 446–449  
sorting collections, 451  
type constructors, 449–450  
UNION, INTERSECT, OVERLAPS, and EXCEPT  
operators, 450  
VALUE modifier, 441–442  
WHERE clause constraints throwing exceptions,  
454–455

**EntityCollections**, **489–491**

EntityContainer **subgroup, 398–399,**  
**411–412**  
EntityDataReader, **456–458**  
EntityDataSource **server control**  
adding a control to ASP.NET Web application  
page, 555–557  
adding a linked DetailsView, 564–565  
binding and formatting a GridView control,  
562–563  
control added to ASP.NET Web application page,  
555–557  
EntityDataSource events, 561–562  
EntityDataSource properties, 557–558  
overview, 4, 554–555, 557  
using the GroupBy property and a  
DropDownList, 563–564  
Where and GroupBy clauses, 559–560

**EntityKey**, **504–505**

**EntityName** **partial classes**  
C# 3.0, 473–475  
overview, 472  
VB 9.0, 475–477

**EntityObject**  
associated entities, 507–508  
selecting and adding new, 551–553  
updating associated, 507–508

**EntityQueries**, **376**

**EntityRef**  
eager loading values to reduce database server  
traffic, 231–233  
substituting for ForeignKey values, 228–230

**entity-relationship diagrams**, **361–362**

**entity-relationship model**, **360–362**

**EntityState**  
logging, 506  
ObjectStateManager and, 504

**EntityType**  
editing properties, 368–369  
filters, 558  
INSERT, UPDATE, AND DELETE functions  
mapped to, 406–407  
Key subgroup, 400  
modifying mapping fields to EntityType  
properties, 53–54  
overview, 503  
Property subelement, 400–401  
subgroups, 400–402, 413

**equality operators**, **142–143**

**equi-joins**, **110–111**

**eSQL** ObjectQueries

C# 3.0, 480  
 merge options and execution times, 482–483  
 overview, 480  
 VB 9.0, 481

**eSqlBlast**  
 Connection page, 434–435  
 Model page, 435–436  
 overview, 434  
 Query page, 436–437  
 Result page, 437–439

**event handlers**  
 for ComboBoxColumns, 543–544  
 deleting dependent records with, 240–242  
 updating association sets with, 545–546

**events, EntityDataSource**, **561–562**

**EXCEPT (eSQL)**, **491**

**Except (QBM)**  
 composing QBM, 491  
 requiring sub-queries, 450  
 as a SQO set operator, 132

**exceptions, subqueries throwing**, **454–455**

**execution times**  
 eSQL ObjectQueries, 483  
 LINQ to Entities queries, 499

**\$expand, ADO.NET query string**, **577**

**expand, ANDS query**, **333**

**explicit loading**, **483**. See also **deferred loading**

**explicit projections, in Entity SQL and Transact-SQL**, **441**

**expression syntax**, **162**

**expression trees**, **27**  
 C# 3.0, 83–84  
 overview, 83  
 query expression trees, 172–176  
 VB 9.0, 86–87  
 VS2008 SP1, 84–86

**expressions, LINQ to Expressions**, **7**

**extension methods**  
 C# 3.0, 72–74  
 LINQ to XML, 270–271  
 overview, 60, 72  
 VB 9.0, 74–75

**extensions, C#**  
 anonymous methods and generic predicates, 75–77  
 anonymous types, 71–72  
 array initializers with object initializers, 70  
 collection initializers, 70–71  
 expression trees and compiled queries, 83–88  
 extension methods, 72–75

implicitly typed local variables, 67–68  
 IQueryables<T> implementations, 88–89  
 lambda expressions, 77–80  
 object initializers, 68–69  
 overview, 66–67  
 standard query operators, 80–83

**extensions, VB**  
 anonymous methods and generic predicates, 75–77  
 anonymous types, 71–72  
 array initializers with object initializers, 70  
 collection initializers, 70–71  
 expression trees and compiled queries, 83–88  
 extension methods, 72–75  
 implicitly typed local variables, 67–68  
 IQueryables<T> implementations, 88–89  
 lambda expressions, 77–80  
 object initializers, 68–69  
 overview, 66–67  
 standard query operators, 80–83

## F

**facets**, **400–401**

**\$filter, ADO.NET query**, **577**

**filter, ANDS query**, **333**

**First( ) operator**, **144–145**

**FirstOrDefault( ) operator**, **144–145**

**FixedLength, SQL Server facet name**, **401**

**FLATTEN operator**, **452**

**Flickr, LINQ to Flickr**, **7**

**foreign-key values**  
 EntityRef substituting for, 228–230  
 SQL Server Foreign Key Relationships, 238–239

**form controls, Windows**. See **Windows form controls, binding to entities**

**Function subelements and subgroups**  
 Function definitions, 404–405  
 Function imports, 405–406  
 INSERT, UPDATE, AND DELETE function mapping to EntityTypes, 406–407  
 overview, 402–404  
 update stored procedure association mapping, 407–408

**functional construction**  
 C# XDocuments, 300–302  
 XML documents, 44, 280–284

**functional languages**, **60–61**

FunctionImportMapping **element**, **418**

# FunctionImports

---

**FunctionImports, 521–522**  
**functions, query, 335**  
**functions, SQL, 30**  
**futures, EF (Entity Framework), 467–468**

## G

### generated classes

C#, 17–19  
examining, 206–210  
VB 9.0, 17, 19–20

### generated entity classes

C# 3.0 and VB 9.0, 470  
EntityName partial classes, 472–477  
ModelNameEntities partial classes,  
471–472  
overview, 470  
serialization, 477–479

### generation operators

Empty, 147  
overview, 147  
Range, 147  
Repeat, 147

### generic predicates, 75–77

### graphs, object

deferred loading of associated objects with  
Load( ) method, 537–538  
deferred loading with the Attach() method,  
539–540  
eager loading of multiple associated objects  
with Include() method, 538–539  
shaping, 535–537

### GridView control, 562–563

Group By  
clauses, 557–560  
displaying customers by country, 563–564  
EntityDataSource property name, 558  
GROUP BY (eSQL), 491  
GroupBy (QBM), 491  
grouping operators, 125–128  
Join and, 164–166

### Group By, with aggregate queries

emulating left outer joins, 168–169  
grouping associated child objects, 161–162  
grouping joined child objects, 163–164  
Join and Group By operations combined,  
164–166  
joins and nested queries compared with,  
166–168

overview, 160–161  
**group joins**  
C# and VB examples, 118–119  
combining Join and GroupBy, 164–166  
compared to nested LINQ queries,  
166–168

creating object graphs, 182–187

overview, 117  
producing hierarchical documents, 289

### grouping

associated child objects, 161–162  
joined child objects, 163–164  
**grouping business documents. See business  
documents, grouping/aggregating**

### grouping operators

ADO.NET Data Services, 333–334, 578  
GroupBy overview, 125–126  
GroupBy with method call syntax, 126–127  
GroupBy with query expression syntax,  
127–128

### groups.child element, 308–312

## H

### Haskell (<http://www.haskell.org>), 60–61

### heterogeneous joins

joining documents and LINQ to SQL or LINQ to  
Object entities, 314–315  
joining documents to insert elements,  
312–313  
lookup operations to add child element groups,  
308–312  
overview, 308

### hierarchical data, 233–236

### hierarchical documents, 289

### hierarchical group join expressions

combining Join and GroupBy with, 164–166  
compared to nested LINQ queries, 166–168

### HTTP 400, 338–340

## I

### IExtendedDataRecord, 456–458

### implicitly typed lambda expression, 79–80

### implicitly typed local variables

C# and VB examples, 67–68  
overview, 60

### imports, Function, 405–406

### IN( ) function, 171–172

`Include()`  
 eager loading with, 485–489, 538–539  
`EntityDataSource`, 558  
`LINQ to Entities` queries, 495–497

**incubator projects, 47**

**index arguments, 104–105**

**index value expressions, 107–108**

`IndexOf( )`, **104–105**

`InDocumentOrder(T)`, in XML extension method, **271**

**inheritance, TPH (table-per-hierarchy)**  
 base and derived class queries, 421–423  
 disambiguate derived object types with base types, 425–427  
 incorrect results from `is`, `Is`, `TypeOf` and `OfType` operators, 423–424  
 overview, 419  
 specifying discriminator column/creating a derived class, 419–421  
 type discrimination in Entity SQL queries, 424–425

**initialization code**  
`C#` example, 98–100  
`VB` example, 100–101

**initializers**  
`array`, 70  
`collection`, 60, 70–71  
`mock object`, 177–182

**initializers, object**  
 with array initializers, 70  
`C#`, 68, 69, 70  
 overview, 60  
`VB`, 69

**INSERT stored procedures**  
 function mapping to `EntityType`s, 406–407  
 inserting a new entity instance, 524–525  
 substituting for dynamic SQL, 224–225

**instantiating `DataContext`, 210–211**

**IntelliSense**  
 enabling for namespaces, 302–304  
 enabling for `VB` queries, 273–275

**Intersect**  
`SQO` set operator, 131–132  
 sub-queries required by, 450

**INTERSECT (eSQL), 491**

**Intersect (QBM), 491**

**Into expression, with VB, 161–162**

**inverted enumeration, 321**

**IOrderedQueryable<T>, 88–89**

**IQueryable<T>, 88–89**

**IS OF operation, for polymorphic queries, 453–454**

**Is operator, incorrect results from, 423–424**

**is operator, incorrect results from, 423–424**

**IsAggregate attribute, Entity SQL, 403**

**IsBuiltIn attribute, Entity SQL, 403**

**IsComposable attribute, Entity SQL, 403**

**ISingleResult<TEntity>, 223–224**

**IsNiladic attribute, Entity SQL, 403**

**J**

**JavaScript Object Notation (JSON)**  
 overview, 598–599

WCF `DataContractJsonSerializer` (DCJS), 477

**Join operators**

combined with `GroupBy`, 164–166  
`GroupJoin`, 117–119  
`Join`, 116–117  
 overview, 115–116

**joined child objects, 163–164**

`join...on` **expressions**  
 code examples, 28–29  
 to create explicit joins, 27–29

**joins, as last resort, 445**

**joins, group**

`C#` and `VB` examples, 118–119  
 combining `Join` and `GroupBy`, 164–166  
 compared to nested LINQ queries, 166–168  
 creating object graphs, 182–187  
 overview, 117  
 producing hierarchical documents, 289

**joins, heterogeneous**

joining documents and LINQ to SQL or LINQ to Object entities, 314–315  
 joining documents to insert elements, 312–313  
 lookup operations to add child element groups, 308–312  
 overview, 308

**JSON (JavaScript Object Notation)**

overview, 598–599

WCF `DataContractJsonSerializer` (DCJS), 477

**K**

**KEY, entity reference, 446–449**

**Key subgroup, EntityType, 400**

## keys

adding members to a collection with, 237  
association, 239–240  
composite primary. See composite primary keys  
entity primary, 360  
natural, 546  
surrogate, 546

## keywords. See also specific keywords

SQOs as, in C# and VB, 93–94  
unclassified, 93  
unsupported in LINQ, 494

# L

## lambda expressions

C# 3.0, 78–79  
overview, 60, 77–78  
VB (8.0 and 9.0), 79–80

## lambda function, 158

## Language Integrated Query. See LINQ (Language Integrated Query)

`Last()` operator, 145  
`LastOrDefault()` operator, 145

## layered mapping, in ADO.NET Entity Framework, 393–394

## lazy evaluation, in SQOs, 82

## lazy loading

associated entities, 338–340  
child objects, 27  
LINQ toSQL and, 214  
SQOs and, 82

## LDAP, LINQ to LDAP, 7

## leaky abstraction, 210

## left outer joins, 168–169

## Let

index value expressions with, 107–108  
as temporary local aggregate variable, 158–159

## libraries

AJAX Client Library. See AJAX Client Library  
.NET 3.5 Client Library. See .NET 3.5 Client Library

## lifetime, `ObjectContext`, 513–514

## LIMIT subclauses, of ORDER BY clause, 452–453

## LIMOG (LINQ in-memory object generator)

creating mock object classes and initializers, 177–178  
creating object graphs, 182, 186, 188  
LINQ to Objects codes and, 96–97

## LINQ (Language Integrated Query)

anonymous methods and generic predicates, 75–77  
anonymous types, 71–72  
array initializers with object initializers, 70  
C# and VB Extensions, 66–67  
collection initializers, 70–71  
compiled queries, 87–88  
copying query results to DataTables, 260–266  
defined, 3  
emerging implementations, 317–318  
expression trees, 83–87  
extension methods, 72–75  
form of basic query, 6–7  
history of, 61–62  
implicitly typed local variables, 67–68  
`IQueryable<T>` and `IOrderedQueryable<T>`, 88–89  
lambda expressions, 77–80  
namespaces, 64–65  
object initializers, 68–69  
overview, 5–8, 63–64  
standard query operators, 80–83  
summary, 89  
third-party implementations, 7  
unsupported keywords, 494

## LINQ in-memory object generator. See LIMOG (LINQ in-memory object generator)

## LINQ Project Sample Query Explorer, 95–96

## LINQ to Active Directory

implementation of, 342–344  
overview, 190, 318

## LINQ to Amazon, 7

## LINQ to DataSets

comparing DataSets and DataContexts, 244–245  
copying query results to DataTables, 260–261  
copying typed DataRows, 262–263  
customizing lookup lists, 253–254  
data binding with `As DataView( )`, 257–260  
defined, 4, 190  
as domain-specific implementation, 193  
features, 246–247  
joining DataSets, 35–37  
navigating entity set associations, 37–38  
overview, 34–35  
processing anonymous types from projections, 263–266  
querying DataTables with, 243–244  
querying typed DataSets, 254–257

querying untyped `DataSets`, 248–252

read-only queries, 247–248

T-SQL, 35

### **LINQ to Entities**

comparing performance, 499

compiling queries, 497–498

conventional queries, 494–495

creating object graphs, 182–187

defined, 4

implementing LINQ to Entities provider, 388–390

`Include()` operator, 495–497

mock collections for testing, 176–177

mock object classes and initializers, 177–182

Out-of-Band SQL updates compared to, 514–515

overview, 38–39

provider role in EF, 363–364

unsupported LINQ keywords, SQOs and overloads, 494

writing queries, 493–494

### **LINQ to Expressions, 7**

#### **LINQ to Flickr, 7**

#### **LINQ to LDAP, 7**

#### **LINQ to NHibernate, 7**

#### **LINQ to Objects**

aggregate operators, 149–153

`AsEnumerable` operator, 133

`AsQueryable` operator, 133–135

C# class definition and initialization code example, 98–100

`Cast` operator, 135–136

concatenation operator: `Concat`, 120–121

conversion operators, 132

data binding with `ToList()` operator, 12–15

`DefaultIfEmpty()` operator, 146

element operators, 144

`ElementAt()` and `ElementAtOrDefault()` operators, 146

equality operators, 142–143

`First()` and `FirstOrDefault()` operators, 144–145

generation operators, 147

grouping operators, 125–128

`GroupJoin` operator, 117–119

index arguments and `IndexOf()`, 104–105

`Join` operators, 115–117

joining documents and, 314–315

`Last()` and `LastOrDefault()` operators, 145

LINQ Project Sample Query Explorer, 95–96

`OfType` operator, 136–138

ordering operators, 121–124

overview, 3, 8–9, 91–92

partitioning operators, 111

projection operators, 105

quantifier operators, 148–149

sample classes for code examples, 96–98

sample queries, 9–12

`Select` operator, 106–108

`SelectMany` operator, 108–111

set operators, 128–132

`Single()` and `SingleOrDefault()` operators, 145–146

`Skip` operator, 112–113

`SkipWhile` operator, 114–115

SQOs as keywords, 93–94

SQOs by group, 92–93

summary, 153–154

`Take` operator, 112

`TakeWhile` operator, 113–115

`To . . .` operators, 138–142

VB class definition and initialization code example, 100–101

`Where` restriction operator, 101–104

### **LINQ to REST. See also Northwind.svc WCF**

#### **Service example**

adopting URIs as a query language, 328–329

C# 3.0 and VB 9.0 queries, 586–587

executing queries, 337–338

as a LINQ implementation, 325–328

overview, 190, 317

### **LINQ to SharePoint**

creating a sample project, 345–346

installing assemblies, 344–345

installing SharePoint 3.0 SDK, 345

overview, 190, 318, 344

translating query expressions, 348–349

writing query code, 346–348

### **LINQ to SQL**

application of (overview), 190, 195–196

binding controls to entity data sources, 21–23

creating mock object classes and initializers, 177–182

creating object graphs, 182–187

domain-specific LINQ implementations, 191–192

vs. EF (Entity Framework), 466–467

emulating joins by navigating entity associations, 24–27

# LINQ to SQL (continued)

---

## **LINQ to SQL (continued)**

emulating SQL functions, modifiers and operators, 30  
examining the generated classes, 16–20  
explicit joins, 27–29  
generating SQL server database with OR designer, 34  
joining documents and, 314–315  
LinqDataSource control, 34  
loading child objects lazily or eagerly, 27  
mapping tables to entities, 15–16  
mock collections for testing, 176–177  
ObjectContext and, 469  
object/relational mapping. See object/relational mapping  
overview, 4, 15  
query pipeline, 212–215  
uploading table data, 30–34

## **LINQ to SQL, as a DAL**

adding, updating and removing objects, 215–219  
detecting and resolving concurrency conflicts, 219–222  
LINQ to SQL query pipeline, 212–215  
moving the LINQ to SQL layer to a middle tier, 225  
overview, 212  
substituting stored procedures, 222–225

## **LINQ to stored XML**

emerging LINQ implementations, 341–342  
overview, 317

## **LINQ to Streams, 7**

### **LINQ to XML**

1:many and many:1 associations, 290–291  
aggregating order details and subtotals by customer, 292–294  
class hierarchy, 270  
composing XML Infosets, 278–280  
document manipulation, 267  
domain-specific LINQ implementations, 193  
functional construction with C# 3.0, 280–284  
grouping elements and aggregating numeric values, 288  
GroupJoin for hierarchical documents, 289  
heterogeneous joins and lookup operations, 308  
implicit vs. explicit typing, 276–278  
inferring a schema and enabling IntelliSense for VB queries, 273–275  
integrating XML into the CLR, 267–269

joining documents, 312–315  
LINQ to XML objects, 39–40  
literal construction with VB 9.0, 284–288  
lookup operations for adding child element groups, 308–312  
overview, 4, 39, 190  
queries, namespaces in C#, 297–298  
querying basic XML Infosets, 271–273  
querying XML documents, 40–44  
querying XML with C#, 269  
summary, 316  
System.Xml.Linq namespace, 269–271  
transforming or creating XML documents, 44–47  
VB axis properties and, 275–276  
Xen for minimizing XML object mismatch, 268  
XML namespaces, 295–297  
XML namespaces in C# 3.0, 297–302  
XML namespaces in VB 9.0, 302–308

## **LINQ to XSD, 44–47**

emerging implementation of, 340–341  
emerging LINQ implementations, 340–341  
history of, 341  
overview, 4, 47–48, 317

## **LINQ4SP, 7**

**LinqDataSource control, 34**  
adding to a page, 226–228  
eager loading EntityRef values for, 231–233  
overview, 4, 226  
substituting EntityRef for ForeignKey values, 228–230

## **LinqDataViews, 257–260**

## **LINQRayTracer, 324–325**

**literal construction, with VB 9.0 in XML, 284–288**

## **literal queries, 304–305**

### **literal XML**

construction with VB 9.0, 284–288  
multiple namespaces transformed with code, 306–308

## **Load() method, 484–485, 537–538**

### **loading, deferred**

of associated entities, 478, 483–489  
Attach() method, 539–540  
Load() method, 484–485, 537–538  
overview, 483

### **loading, eager**

of associated entities, 478–479  
of child objects, 27

- EntityRef values to reduce server traffic, 231–233
- `Include()` method, 485–489, 538–539
- overview, 483
- loading, lazy**
- associated entities, 338–340
  - child objects, 27
  - LINQ toSQL and, 214
  - SQOs and, 82
- local names, XML.** *See* **XML namespaces**
- local variable type inference**
- C# and, 67
  - implicitly typed, 67–68
  - overview, 60
  - VB and, 68
- logical operators, ADO.NET Data Services, 333–334, 578**
- LongCount, as an aggregate operator, 150**
- lookup operations**
- C# 3.0 functional construction, 309–310
  - customizing lookup lists, 253–254
  - overview, 308–309
  - VB 9.0 functional construction, 311–312
  - VB 9.0 literal XML, 310–311
- ## M
- Many:1 associations**
- in business documents, 290–291
  - navigation properties, 442–443
- mapping.** *See also* **object/relational mapping**
- associations, 407–408
  - attribute-centric, 20, 210
  - EDM Designer and, 53–54
  - EDM terminology definition, 360
  - files with `SqlMetal.exe`, 200–203
  - function mapping to `EntityType`s, 406–407
  - from physical to conceptual schema, 50–52
  - tables to entity sets, 197–200
- Mapping (MSL content)**
- `AssociationSetMapping` subgroup, 417–418
  - `EntitySetMapping` subgroup, 415–417
  - `FunctionImportMapping` element, 418
  - overview, 414
- mapping files**
- `*.dbml` and `*.xml` files, 203–206
  - generating with `SqlMetal.exe`, 200–203
  - role as EF (Entity Framework) component, 363–364
- Mapping Schema Language.** *See* **MSL (Mapping Schema Language)**
- Mapping subgroup, AssociationSet, 417–418**
- mappings group, 373–375**
- mashup, Web, 326**
- math functions, ADO.NET Data Services, 335, 579**
- Max operator, 151**
- MaxLength facet, SQL Server, 401**
- members, adding to collections, 237**
- merge options**
- eSQL ObjectQueries, 482
  - updating associated EntitySets or `EntityObjects`, 507–508
- \$metadata, ADO.NET query string option, 577**
- MetadataWorkspace**
- C# 3.0, 428–429
  - `ObjectContext` and, 383–384
  - traversing, 427–428
  - VB 9.0, 429–431
- method calls**
- `GroupBy` with, 126–127
  - index arguments and, 104–105
  - numerical operators and, 157–158
  - translating query expressions to, 348–349
- Microsoft Entity Data Platform, 391–393**
- Microsoft Synchronization Framework, 5**
- middle tier, LINQ to SQL layer, 225**
- Min operator, 151**
- mocking collections**
- creating mock object classes and initializers, 177–182
  - creating object graphs, 182–187
  - testing LINQ to SQL and LINQ to Entities, 176–177
- model, entity-relationship, 360–361**
- Model page, for eSqlBlast, 435–436**
- modeling a complex type**
- modifying the CSDL section, 529
  - modifying the MSL section, 530
  - overview, 529
  - reusing a `ComplexType` definition, 530–531
- ModelName.edmx files, analyzing**
- `ConceptualModels` group, 371–373
  - `mappings` group, 373–375
  - `StorageModels` group, 370–371
- ModelNameEntities partial classes**
- C# 3.0, 471
  - overview, 471
  - VB 9.0, 472

## models, conceptual

- Association subgroup, 414
- EntityContainer subgroup, 411–412
- EntityType subgroup, 413
- group, 371–373
- overview, 411
- models, Data Model, 521–522**
- models, storage**
  - association subgroups, 409–411
  - EntityContainer subgroup, 398–399
  - EntityType subgroups, 400–401
  - Function subelements and subgroups. See
    - Function subelements and subgroups
  - overview, 397
  - Product EntityType subgroup, 402
  - scanning StorageModels group, 370–371
- modifications, collection, 236–237**
- modifier, VALUE, 441–442**
- modifiers, SQL, 30**
- MSL (Mapping Schema Language)**
  - AssociationSetMapping subgroup, 417–418
  - EntitySetMapping subgroup, 415–417
  - FunctionImportMapping element, 418
  - mapping physical to the conceptual schema, 50–52
  - modification in modeling a complex type, 530
  - overview, 414
- multiple associated objects, 538–539**
- multiple namespaces**
  - transforming with literal XML code, 306–308
  - XDocuments constructed with, 300–302
  - XML literal queries, 304–305
- multiple Select projection expression, 107**
- MULTISET type constructor, 449–450**

# N

## namespaces

- C# expanded, 298–299
- IntelliSense for, 302–304
- LINQ to XML, 297–299
- .NET Framework 3.5, 64–65
- System.Xml.Linq, 269–271
- VB 9.0, 302–308
- namespaces, multiple**
  - transforming documents containing, 306–308
  - XDocuments constructed with, 300–302
  - in XML literal queries, 304–305
- namespaces, XML**
  - enabling IntelliSense for namespaces, 302–304
- examples, 295–296
- functionally constructing XDocuments, 300–302
- multiple namespaces in XML literal queries, 304–305
- namespaces in C# LINQ to XML queries, 297–298
- overview, 295, 302
- removing expanded namespaces, 298–299
- transforming documents with multiple namespaces, 306–308

## natural keys, 546

## NAVIGATE, 445–446

### navigation properties

- many:one, 442–443
- one:many, 444–445

### nested queries

- NAVIGATE as a complex substitute for, 445–446
- one:many navigation, 444–445
- sorting collections requiring, 451

## .NET 3.5 Client Library

- adding new entities, 592–594
- batching changes, 597
- deleting entities, 596–597
- executing batch queries with
  - DataServiceRequests, 589–591
- executing DataServiceQuery, 585–586
- executing LINQ to REST queries, 586–587
- executing queries from Windows Forms clients, 583
- executing query-string URLs from clients, 583–585
- managing optimistic concurrency conflicts, 597
- overview, 581–583
- returning associated entities, 587–589
- updating entities, 594–595

## .NET BinarySerializer, 477

## .NET Framework 3.5 namespaces, 64–65

## .NET XmlSerializer, 477

## Next Generation Data Access, 58

## NHibernate, LINQ to NHibernate, 7

## Nodes (T), in XML extension method, 271

## Northwind.svc WCF Service example

- experimenting with URI query syntax, 329–332
- logical, arithmetic and grouping operators, 333–334
- query options, 333
- string, date, math, and type query functions, 335
- in XML (Atom 1.0), 332

NoTracking, **merge option**, 482  
**NQueens**, PLINQ, 324  
**Nullable facet**, SQL Server, 401  
**numeric values of business documents**. See  
 business documents, grouping/aggregating  
**numerical operators**, 157–158  
**NwindServicesClient services requests**  
 executing LINQ to REST queries, 337–338  
 executing URI queries, 337  
 overview, 336  
 preventing HTTP 400, 338–340

## O

**object addition operations, for updates**, 549–550  
**Object data sources, binding Windows forms controls to entities**  
 overview, 534–535  
 persisting changes to the data store, 553–554  
 selecting active top-level/associated entity instances, 540–543  
 setting composite primary-key members. See composite primary keys  
 using unbound ComboBoxes, 543–546  
**object deletion operations, for updates**, 549–550  
**object graphs**  
 creating with GroupJoin expressions, 182–187  
 deferred loading of associated objects with Load( ) method, 537–538  
 deferred loading with the Attach() method, 539–540  
 eager loading of multiple associated objects with Include() method, 538–539  
 instantiating, 210–211  
 overview, 155  
 shaping, 535–537  
**object initializers**  
 with array initializers, 70  
 C#, 68, 69, 70  
 overview, 60  
 VB, 69  
**object layer class file**, 365–367  
**Object Services**  
 compiling LINQ to Entity queries, 497–499  
 composing ObjectQueries, 491–492  
 conventional LINQ to Entities queries, 494–495  
 deferred loading, 484–485  
 eager loading, 485–489  
 EF roles, 363–364

entity class serialization, 477–479  
 EntityName partial class, 472–477  
 eSQL ObjectQueries, 480–483  
 generated entity classes and, 470  
 Include() operator in LINQ to Entity queries, 495–497  
 LINQ to Entities provider, 388–390  
 ModelNameEntities partial class, 471–472  
 ObjectContext, 382–384  
 ObjectQueries with Entity SQL, 384–387  
 ObjectQueries with QBM, 387–388  
 ordering and filtering associated EntityCollections, 489–491  
 ordering/filtering EntityCollections, 489–491  
 overview, 381–382, 469  
 parameterizing ObjectQueries, 500  
 performance of LINQ to Entity queries, 499  
 summary, 500–501  
 unsupported LINQ keywords, SQOs and overloads, 494  
 writing LINQ to Entities queries, 493–494  
**object types, derived**, 425–427  
**ObjectContext**  
 creating ObjectQuery and, 382  
**Object Services** and, 469  
**ObjectStateManager** and children of, 503–505  
 optimizing the lifetime of, 513–514  
**ObjectDataSource**, 226  
**ObjectQuery**  
 composing with QBM, 387–388, 491–492  
 creating ObjectContext and, 382  
 writing with Entity SQL, 384–387  
**object/relational mapping**  
 examining the generated classes, 206–210  
 generating partial entity classes/mapping files, 200–203  
 instantiating DataContext, 210–211  
 mapping tables to entity sets, 197–200  
 overview, 197  
 working with \*.dbml and \*.xml mapping files, 203–206  
**objects. See also LINQ to Objects**  
 adding, updating, and deleting, 215–219, 367–368  
 child, 161–163  
 code examples, 56–57  
 deletion/addition operations for updating, 549–550

## **objects. See also LINQ to Objects (continued)**

domain, 34  
graphs. See object graphs  
initializers, 60, 68–70  
LINQ to XML, 39–40  
materializing object context, 56–57  
persistence, 195–196  
relationship to resources, 395  
XML/object mismatch, 268

**objects, associated**  
deferred loading with `Attach()` method, 539–540  
deferred loading with `Load()` method, 537–538  
eager loading with `Include()` method, 538–539  
equi-joins of, 110–111  
grouping, 161–162  
`SelectMany` implementation, 108–109

**ObjectStateManager**  
methods and event, 504  
understanding, 503–505  
in working with `ObjectContext`, 384

**OFTYPE (eSQL), 491**

**OfType (eSQL)**  
as a conversion operator, 136–138  
incorrect results from, 423–424  
as type operator for polymorphic queries, 453–454

**OfType (QBM), 491**

**one:many, navigation properties, 444–445**

**operators. See also specific SQOs**  
code examples, 30  
emulating SQL, 30  
operator groups list, 92–93

**operators, aggregate**  
`Aggregate`, 152–153, 159  
`Count` and `LongCount`, 150  
`Let` variable, 158–159  
method call syntax, 157–158  
`Min`, `Max`, `Sum`, and `Average`, 151  
overview, 149, 156  
query output formatting, 159–160

**operators, arithmetic, 578**

**operators, comparison, 578**

**operators, compound OR, 169–171**

**operators, concatenation, 120–121**

**operators, conversion**  
`AsEnumerable`, 133  
`AsQueryable`, 133–135

`Cast`, 135–136  
`OfType`, 136–138  
To...operators. See To...operators

**operators, element**  
`DefaultIfEmpty`, 146  
`ElementAt` and `ElementOrDefault`, 146  
`First` and `FirstOrDefault`, 144–145  
`Last` and `LastOrDefault`, 145  
overview, 144  
`Single` and `SingleOrDefault`, 145–146

**operators, equality, 142–143**

**operators, generation**  
`Empty`, 147  
`Range`, 147  
`Repeat`, 147

**operators, grouping**  
ADO.NET Data Services and, 333–334, 578  
method call syntax for, 126–127  
overview, 125–126  
query expression syntax for, 127–128

**operators, Join**  
combined with `GroupBy`, 164–166  
`GroupJoin`, 117–119  
`Join`, 116–117  
overview, 115–116

**operators, logical, 333–334, 578**

**operators, ordering**  
examples, 123–124  
`OrderByDescending`, 122  
overview, 121–122  
`ThenBy`, 122  
`ThenByDescending`, 123

**operators, partitioning**  
overview, 111  
`Skip`, 112  
`Skip/Take` example, 112–113  
`SkipWhile`, 114  
`SkipWhile/TakeWhile` example, 114–115  
`Take`, 112  
`TakeWhile`, 113–114

**operators, project**  
overview, 105  
`Select`, 106–108  
`SelectMany`, 108–111

**operators, quantifier**  
`All`, 148–149  
`Any`, 148  
`Contains`, 149  
overview, 148

**operators, restriction, 101–105**  
 compound Where expressions, 102–104  
 method calls with index arguments and use of  
   IndexOf, 104–105  
 simple Where expressions, 102  
 Where overview, 101

**operators, set**  
 Distinct, 129  
 Except, 132  
 Intersect, 131–132  
 overview, 128  
 Union, 129–131

**operators, standard query, 80–83**

**operators, To...**  
 overview, 138  
 ToArray, 138  
 ToDictionary, 139–140  
 ToList, 139  
 ToLookup, 141–142

**operators, type, 453–454**

**optimistic concurrency management**  
 changing EntitySets and, 597  
 code for implementing, 517–520  
 enabling for entity properties, 516–517  
 overview, 515–516

**O/R Designer**  
 editing \*.dbml files in, 203–204  
 generating a SQL server database with, 34  
 mapping tables to entity sets, 15–16,  
   197–200

**OR operators, compound, 169–171**

**ORDER BY clause, 452–453**

**ORDER BY (eSQL), 491**

**Order\_Details, aggregating, 290–291**

**OrderBy (QBM), 491**

**OrderBy, ADO.NET Data Services, 333**

**\$orderby, ADO.NET query string option, 577**

**OrderBy property, EntityDataSource, 558**

**OrderByDescending, 122**

**ordering operators**  
 C# 3.0 example, 123–124  
 OrderByDescending, 122  
 overview, 121–122  
 ThenBy, 122  
 ThenByDescending, 123  
 VB 9.0 example, 124

**OrdersSubtotals, aggregating, 290–291**

**Out-of-Band SQL, 514–515**

**OVERLAPS set operators, 450**

**OverwriteChanges, 482**

**P**

**pages, eSqlBlast**  
 Connection page, 434–435  
 Model page, 435–436  
 Query page, 436–437  
 Result page, 437–439

**paging, SKIP and LIMIT subclauses and, 452–453**

**Parallel Computing Initiative (PCI), 319**

**Parallel LINQ. See PLINQ (Parallel LINQ)**

**Parameter Source types, 560**

**parameterized queries**  
 execution of, 460–461  
 ObjectQuery, 500  
 ParameterTypeSemantics attribute,  
   Entity SQL, 403

**partial entity classes**  
 EntityName, 472–477  
 generating with SqlMetal.exe, 200–203  
 ModelNameEntities, 471–472

**partitioning operators**  
 overview, 111  
 Skip, 112  
 Skip/Take example, 112–113  
 SkipWhile, 114  
 SkipWhile/TakeWhile example, 114–115  
 Take, 112  
 TakeWhile, 113–114

**PCI (Parallel Computing Initiative), 319**

**performance**  
 Entity SQL queries and, 459  
 LINQ to Entities updates, 514–515  
 Out-of-Band SQL updates, 514–515  
 query expression trees for improving,  
   172–176

**Persistence Ignorance (PI), 20, 210, 390–391**

**pessimistic concurrency management, 515**

**physical layer, mapping to conceptual**  
 adding, updating, and deleting the model's  
   objects, 367–368  
 creating the XML mapping files and object layer  
   class file, 365–367  
 editing EntityType and AssociationSet  
   properties, 368–369  
 overview, 50–52, 365

**PI (Persistence Ignorance), 20, 210, 390–391**

**pipelined processing, 321**

# PLINQ (Parallel LINQ)

---

## **PLINQ (Parallel LINQ), 5**

overview, 191, 317–319  
processing queries, 321–322  
programming with, 319–321  
running the samples, 322–325

## **pluralization, 16**

## **polymorphic queries, 453–454**

Precision facet, SQL Server, 401

## **predicates, generic, 75–77**

## **prefixes, entity alias, 440–441**

Preserve Seconds facet, SQL Server, 401

PreserveChanges, merge option, 482

## **primary keys, adding members to a collection with, 237**

## **primary keys, composite**

detecting attempts to change values, 546–549  
enforcing object deletion and addition, 549–550  
overview, 546  
selecting/adding EntityObject and AssociationSet, 551–553

## **procedures, stored. See stored procedures**

Product EntityType subgroup, storage models and, 402

Product stored procedures, 404–405

programming, DataContext. See DataContext

programming, with PLINQ, 319–321

## **project operators**

overview, 105  
Select, 106–108  
SelectMany, 108–111

## **projection expression**

multiple Select, 107  
simple Select, 106–107

## **projections**

anonymous. See anonymous types  
explicit, 441

## **properties**

EntityDataSource, 557–558  
EntityType subgroups, 400–401  
timestamp, 527–528  
values, 560

# Q

## **QBM<sub>s</sub> (Query Builder Methods), 387–388, 491–492**

## **quantifier operators**

All, 148–149  
Any, 148

Contains, 149

overview, 148

## **queries**

ADO.NET Data Services options, 333  
batch, 589–591  
conventional, 494–495  
copying results to DataTables, 260–266  
derived classes, 421–423  
Dim Query, 6  
expressions. See query expressions  
formatting output of, 159–160  
LINQ. See LINQ (Language Integrated Query)  
polymorphic, 453–454  
processing in PLINQ, 321–322  
query language, 328–329  
standard query operators. See SQOs (standard query operators)  
typed DataSets, 253–254  
URI query syntax, 329–332  
var, 6  
XML namespaces, 297–298, 304–305

## **queries, advanced**

aggregate operators. See aggregate operators  
Contains() SQO. See Contains() SQO  
GroupBy, with aggregate queries. See Group By, with aggregate queries  
mocking collections for testing projects. See mocking collections  
overview, 155  
query expression trees, 172–176  
summary, 187–188

## **queries, base classes**

disambiguate derived object types with base types, 425–427  
incorrect results from is, Is, Typeof and OfType operators, 423–424  
overview, 421–423  
type discrimination in Entity SQL queries, 424–425

## **queries, compiled**

code examples, 87–88  
LINQ to Entities, 497–499  
overview, 87

## **queries, Entity SQL**

as client views. See Entity SQL, queries as client views  
against EntityClient. See Entity SQL, queries against EntityClient

## **queries, from Windows Form clients**

executing a DataServiceQuery, 585–586

- 
- executing a query-string URI, 583–585  
executing batch queries with  
  DataServiceRequests, 589–591  
executing LINQ to REST queries, 586–587  
returning associated entities, 587–589
- queries, nested**  
NAVIGATE as a complex substitute for,  
  445–446  
one:many navigation, 444–445  
sorting collections requiring, 451
- queries, parameterized**  
execution of, 460–461  
ObjectQuery, 500
- queries, read-only LINQ to DataSet**  
customizing lookup lists, 253–254  
querying typed DataSets, 254–257  
querying untyped DataSets, 248–252
- queries, URI**  
adopting URI as a query language, 328–329  
executing, 337  
executing a query-string URI, 583–585  
experimenting with URI query syntax, 329–332
- queries, XML Infosets**  
C# 3.0 example, 271–272  
implicit v. explicit content typing, 276–278  
inferring a schema/enabling IntelliSense,  
  273–275  
overview, 271  
taking advantage of VB 9.0 axis properties,  
  275–276  
VB 9.0 example, 272–273
- Query Builder Methods (QBMIs), 387–388, 491–492**
- query code, 346–348**
- query expressions**  
advanced. See advanced queries  
overview, 80–83  
syntax of, 127–128  
translating, 348–349  
trees, 172–176
- query language, 328–329**
- Query page, for eSqlBlast, 436–437**
- query pipeline, in LINQ to SQL, 212–215**
- query string options, 576–580**  
comparison, logical, arithmetic, and grouping  
  operators, 578  
DateTime, math and type functions, 579–580  
option descriptions, 577  
overview, 576
- query-string URI, 583–585**
- R**
- Range **operator, 147**
- ray tracing, 322**
- read-only LINQ to DataSet queries**  
customizing lookup lists, 253–254  
querying typed DataSets, 254–257  
querying untyped DataSets, 248–252
- RectangleUnion, PLINQ, 324**
- REF, entity reference, 446–449**
- relationship sets, 360**
- relationships**  
EDM terminology definition, 360  
entities and, 196  
entity-relationship model, 360–362
- Remove, in XML extension method, 271**
- removing objects, in LINQ to SQL, 215–219**
- Repeat operator, 147**
- Representational State Transfer. See REST (Representational State Transfer)**
- resources, relationship to objects, 395**
- REST (Representational State Transfer).**  
*See also LINQ to REST*  
architecture, 570  
definition, 325
- restriction operator**  
compound Where expressions, 102–104  
method calls with index arguments and  
  use of IndexOf, 104–105  
simple Where expressions, 102  
Where overview, 101
- Result page, for eSqlBlast, 437–439**
- roles, EDM, 360**
- ROW**  
entity reference, 446–449  
type constructors creating, 449–450
- S**
- schema**  
for \*.dbml and \*.xml files,  
  202–206  
enabling IntelliSense, 273–275  
OR Designer generated, 34  
physical and conceptual, 49–52  
stored procedures and, 223  
XML, 395–397  
XML documents, 42–47

## **schema, database**

\*.dbml and \*.xml files and, 202–206

OR Designer generated, 34

physical and conceptual, 49–52

XML documents, 42–47

## **schema, mapping physical to conceptual**

adding, updating, and deleting objects, 367–368

creating the XML mapping files and object layer class file, 365–367

editing `EntityType` and `AssociationSet` properties, 368–369

overview, 365

### **SELECT (eSQL), 491**

#### **Select (QBM), 491**

#### **Select project operators**

index value expressions with the `Let` keyword, 107–108

multiple select projection expression, 107

overview, 106

simple select projection expression, 106–107

#### **Select projection expression, 106–107**

#### **Select property, EntityDataSource, 558**

#### **SELECT stored procedure, 223–224**

#### **SELECT VALUE (eSQL), 491**

#### **SelectedIndexChanged event handler, 545–546**

#### **SelectMany project operator**

basic implementation for associated objects, 108–109

overload for equi-joins of associated objects, 110–111

overview, 108

#### **SelectValue (QBM), 491**

#### **SequenceEqual operator, 142–143**

#### **serialization, entity class, 477–479**

#### **server controls, ASP.NET. See**

`EntityDataSource` server control

#### **service operations, invoking in ADO.NET Data Services, 580–581**

#### **services requests, processing. See**

`NwindServicesClient` services requests

#### **set operators**

`ANYELEMENT` and `FLATTEN` work on collections, 452

`Distinct`, 129

`Except`, 132

`Intersect`, 131–132

overview, 128

sorting collections returned by, 451

`Union`, 129–131

## **sets, entity**

adding `FunctionImports` to populate, 521–522

adding new entities, 592–594

associated entities and, 507–508

batching changes, 597

data retrieval stored procedures assigned to, 405–406

deleting entities, 596–597

deleting members from collections, 238–242

EDM terminology definition, 360

elements or subgroups, 398

`EntitySetNameMapping`, 415–417

`EntitySetName`, 558

managing optimistic concurrency conflicts, 597

mapping tables to, 197–200

updating, 594–595

updating associated, 507–508

## **SharePoint 3.0 SDK, 345. See also LINQ to SharePoint**

### **SharePoint Mapping Language (SMPL), 345**

#### **Single operator, 145–146**

#### **SingleOrDefault operator, 145–146**

#### **SKIP (eSQL), 491**

#### **Skip (QBM)**

C# and VB examples, 112–113

composing QBMs, 491

partitioning operator, 112

#### **skip, ADO.NET Data Services, 333**

#### **SKIP subclauses, ORDER BY clause, 452–453**

#### **SkipWhile**

C# and VB examples, 114–115

partitioning operator, 114

#### **Slambda, in Entity Framework futures, 466–467**

### **SMPL (SharePoint Mapping Language), 345**

#### **sorting collections, 451**

### **SQL. See also LINQ to SQL**

`IN ( )` function, 171–172

functions, modifiers and operators, 30

overview, 7, 61

`Where` clauses, 169–171

## **SQL Server**

cascading deletions, 238–239

facets, 401

OR Designer generated database, 30–34

### **SQL Server Compact (SSCE), 5, 461–463**

### **SQL Server (Express), 462–463, 494**

`SqlMetal.exe`, 200–203

**SQOs (Standard Query Operators).** See also **specific SQOs**  
 advanced query operators/expressions. See advanced queries  
 C# 3.0 expressions, 80–82  
 by group, 92–93  
 as keywords in C# 3.0 and VB 9.0, 93–94  
 lazy and eager evaluation, 82  
 overview, 80  
 unsupported, 494  
 VB 9.0 expressions, 83

**SSCE (SQL Server Compact),** 5, 461–463

**SSDL (Store Schema Definition Language),** 370–371. See also **StorageModels (SSDL content)**

**Standard Query Operators.** See **SQOs (standard query operators)**

**stop-and-go processing,** 321

**StorageModels (SSDL content)**  
 association subgroups, 409–411  
 EntityContainer subgroup, 398–399  
 EntityType subgroups, 400–401  
 Function subelements and subgroups. See Function subelements and subgroups  
 overview, 397  
 Product EntityType subgroup, 402  
 scanning StorageModels group, 370–371

**Store Schema Definition Language (SSDL),** 370–371. See also **StorageModels (SSDL content)**

**stored procedures**  
 CRUD. See **CRUD** (create, retrieve, update, and delete)  
 data retrieval to an `EntitySet`, 405–406  
 entity instances deleted with, 528  
 Function definitions for, 404–405  
`INSERT`, `UPDATE` AND `DELETE` procedures, 224–225  
 inserting a new entity instance, 524–525  
 population of associations with, 410–411  
`SELECT` procedure, 223–224  
 substituting for dynamic SQL, 222–225  
 update association mapping, 407–408

**StoreGenerated Pattern facet, SQL Server,** 401

`StoreOriginal ValueInViewState` **property**, `EntityDataSource`, 558

**string query functions,** 335

**structs, anonymous,** 269

**subclauses,** `LIMIT`, 452–453

**subclauses, `SKIP`,** 452–453

**subelement, `Property`,** 400–401

**subqueries**  
 throwing exceptions, 454–455  
`UNION`, `INTERSECT`, `OVERLAPS`, and `EXCEPT` require, 450  
`Sum` **operator**, 151

**surrogate keys**  
 application of, 546  
 autoincrementing primary key, 237

**Synchronization Framework, Microsoft,** 5

**syntactic sugar,** 159

**syntax**  
 aggregate operators, 157–158  
 C# grouping expression, 162  
 dot-notation, 442–443  
 expression syntax with `Let`, 158–159  
`GroupBy` with method call, 126–127  
`GroupBy` with query expression, 127–128  
 method call with numerical operators, 157–158

`System.Xml.Linq` **namespace**, 269–271

## T

**table data**  
 C# 3.0, 31–32  
 updating, 30–32  
 VB 9.0, 32–34

**table-per-hierarchy inheritance.** See **TPH (table-per-hierarchy) inheritance**

**tables**  
 data table, 30–34  
 mapping to entities, 15–16, 197–200

**Take** **operator**, 112–113

**TakeWhile** **operator**, 113–115

**temporary local aggregate variable,** 158–159

**test client, AJAX,** 599–602

**testing LINQ to SQL and LINQ to Entities.** See **mocking collections**

**ThenBy** **operator**, 122

**ThenByDescending** **operator**, 123

**third-party LINQ implementations,** 7, 317–318, 342–344

**tier, middle,** 225

**timestamp property, concurrency management and,** 527–528

**ToArray** **operator**, 138

**ToDictionary** **operator**, 139–140

# ToList operator

---

## ToList operator

LINQ data binding with, 12–15  
overview, 139

## ToLookup operator, 141–142

### To . . . operators

overview, 138  
ToArray, 138  
ToDictionary, 139–140  
ToList, 139  
ToLookup, 141–142

### Top (QBM), 491

### \$top, ADO.NET query string option, 577

### TOP AND LIMIT (eSQL), 491

## top-level entity instances, 540–543

## TPH (table-per-hierarchy) inheritance

base and derived class queries, 421–423  
disambiguate derived object types with base types, 425–427  
incorrect results from is, Is, TypeOf and OfType operators, 423–424  
overview, 419

specifying discriminator column/creating a derived class, 419–421  
type discrimination in Entity SQL queries, 424–425

## Transact-SQL. See Entity SQL/Transact-SQL comparison

### TREAT operator, 453–454

### tuples, 71–72, 360

### type discrimination, in Entity SQL queries, 424–425

### type functions, in ADO.NET Data Services, 579

### type operators, for polymorphic queries, 453–454

### type query functions, in ADO.NET Data Services, 335

### typed DataRows, 262–263

### typed DataSets

- joining, 35–36
- querying, 254–257

### TypeOf operator, 423–424

## types, anonymous

C# 3.0, 71  
DataSets, 263–266  
overview, 60  
VB 9.0, 71–72

## types, base

disambiguate derived object types with base types, 425–427

incorrect results from is, Is, TypeOf and OfType operators, 423–424  
overview, 421–423  
type discrimination in Entity SQL queries, 424–425

## types, complex

modeling, 529–531  
overview, 528  
reusing a ComplexType definition, 530–531  
working with, 528–531

## typing, explicit, 276–278

## typing, implicit

of element and attribute content, 276–278  
of lambda expression, 79–80  
of local variables, 60, 67–68

# U

## unbound ComboBoxes, to specify associations

adding event handlers, 543–544  
updating association sets, 545–546

## unclassified keywords, 93

### Unicode facet, SQL Server, 401

### UNION (eSQL), 491

### Union (QBM), 129–131, 491

### UNION ALL (eSQL), 491

### UNION set operators, 450

### UnionAll (QBM), 491

## untyped DataSets

joining, 36–37  
querying, 248–252

## UPDATE stored procedures

function mapping to EntityTypes, 406–407  
substituting for dynamic SQL, 224–225  
updating an entity instance and managing concurrency with original values, 525–527  
updating an entity instance and managing concurrency with timestamp property, 527–528

## updates

enforcing object deletion and addition operations for, 549–550  
LINQ to Entities vs. Out-of-Band SQL, 514–515  
LINQ to SQL, 215–219  
transacting, 508  
validating, 510–513

**URI queries**

- adopting URI as a query language, 328–329
- executing, 337
- executing a query-string URI, 583–585
- experimenting with URI query syntax, 329–332
- `UserDeletingRows`, **dependent records and**, **240–242**

**V****validating data additions and updates, 510–513**

`$value`, **ADO.NET query string option**, **577**

**VALUE modifier, Entity SQL**, **441–442**

**value objects**, **528**

**values**

- aggregation of child entity values, 158
- composite primary key, 546–549
- concurrency management with, 525–527
- value-type association keys**, **239–240**
- var query**, **6**
- variables, implicitly typed local**, **60, 67–68**
- VB 9.0**
  - axis properties of, 275–276
  - inferring a schema for a XML document, 273–275
  - literal XML construction with, 284–288
  - PLINQ, 324–325
  - XML namespaces in, 302–305

**VB extensions**

- anonymous methods and generic predicates, 75–77
- anonymous types, 71–72
- array initializers with object initializers, 70
- collection initializers, 70–71
- expression trees and compiled queries, 83–88
- extension methods, 72–75
- implicitly typed local variables, 67–68
- `IQueryable<T>` implementations, 88–89
- lambda expressions, 77–80
- object initializers, 68–69
- overview, 66–67
- standard query operators, 80–83

**views, client**

- EF components, 363–364
- executing Entity SQL queries as, 376–381
- working with, 375

**VS2008 SP1 ExpressionTreeVisualizer, in SQOs, 84–86****W****WCF (Windows Communication Framework) Service templates, 4**

**WCF DataContractJsonSerializer (DCJS)**, **477**

**WCF DataContractSerializer (DCS)**  
entity class serialization with, 477  
XML from, 478, 479

**Web service, creating and consuming in a browser, 571–572**

- WHERE (eSQL)**
  - clause constraints, 454–455
  - composing QBM, 491
  - compound, 102–104
  - method calls with index arguments, 102–104
  - simple, 102
  - SQL clauses, 169–171

**Where (QBM), 491**

**Where clauses, in EntityDataSource, 557–560**

**Where expressions**

- compound, 102–104
- restriction operators, 102

**Where property, EntityDataSource, 558**

**Windows Communication Framework (WCF) Service templates, 4****Windows Form clients, queries from**

- executing a `DataServiceQuery`, 585–586
- executing a query-string URI, 583–585
- executing batch queries with `DataServiceRequests`, 589–591
- executing LINQ to REST queries, 586–587
- returning associated entities, 587–589

**Windows form controls, binding to entities**

- adding members to collections, 237
- autogenerating hierarchical data editing form, 233–236
- collection modifications, 236–237
- composite primary-key members. See composite primary keys
- deferred loading of associated objects with `Load()` method, 537–538
- deferred loading with the `Attach()` method, 539–540
- deleting members from a collections, 238–242
- eager loading of multiple associated objects with `Include()` method, 538–539
- entity edits, 236–237
- overview, 233, 534–535

## **Windows form controls, binding to entities (continued)**

persisting changes to the data store, 553–554  
persisting entity edits/collection modifications,  
  236–237  
selecting active top-level/associated entity  
  instances, 540–543  
shaping object graphs, 535–537  
using unbound `ComboBoxes`, 543–546

## **WordScrambler, PLINQ, 324**

# X

## **XDocuments, 300–302**

### **Xen, 268**

## **XML. See also LINQ to XML**

literal queries, 304–305  
mapping files, 365–367  
`Northwind.svc` WCF Service in, 332  
object mismatch, 268  
objects, 39–40  
schema, 395–397  
transforming documents with literal code,  
  306–308  
from `WCF DataContractSerializer` (DCS),  
  478, 479

## **XML, integrating into the CLR**

minimizing XML/object mismatch, 268  
overview, 267–268  
querying XML with C#, 269

## **XML documents**

C# 3.0, 40–41, 42, 44–45  
querying, 40–44  
transforming or creating, 44–47, 306–308  
VB 9.0, 41–42, 43, 45–47

## **XML editor, 205–206**

\* `.xml` files, 205–206

## **XML Infosets**

C# 1.0 example, 279  
functional construction with C# 3.0,  
  280–284  
literal construction with VB 9.0, 284–288  
overview, 278–279  
VB 7.0 example, 280

## **XML Infosets, querying**

C# 3.0 example, 271–272  
implicit v. explicit content typing, 276–278  
inferring a schema/enabling IntelliSense,  
  273–275  
overview, 271  
taking advantage of VB 9.0 axis properties,  
  275–276  
VB 9.0 example, 272–273

## **XML namespaces**

examples, 295–296  
overview, 295  
transforming documents with multiple  
  namespaces, 306–308

## **XML namespaces, in C# 3.0**

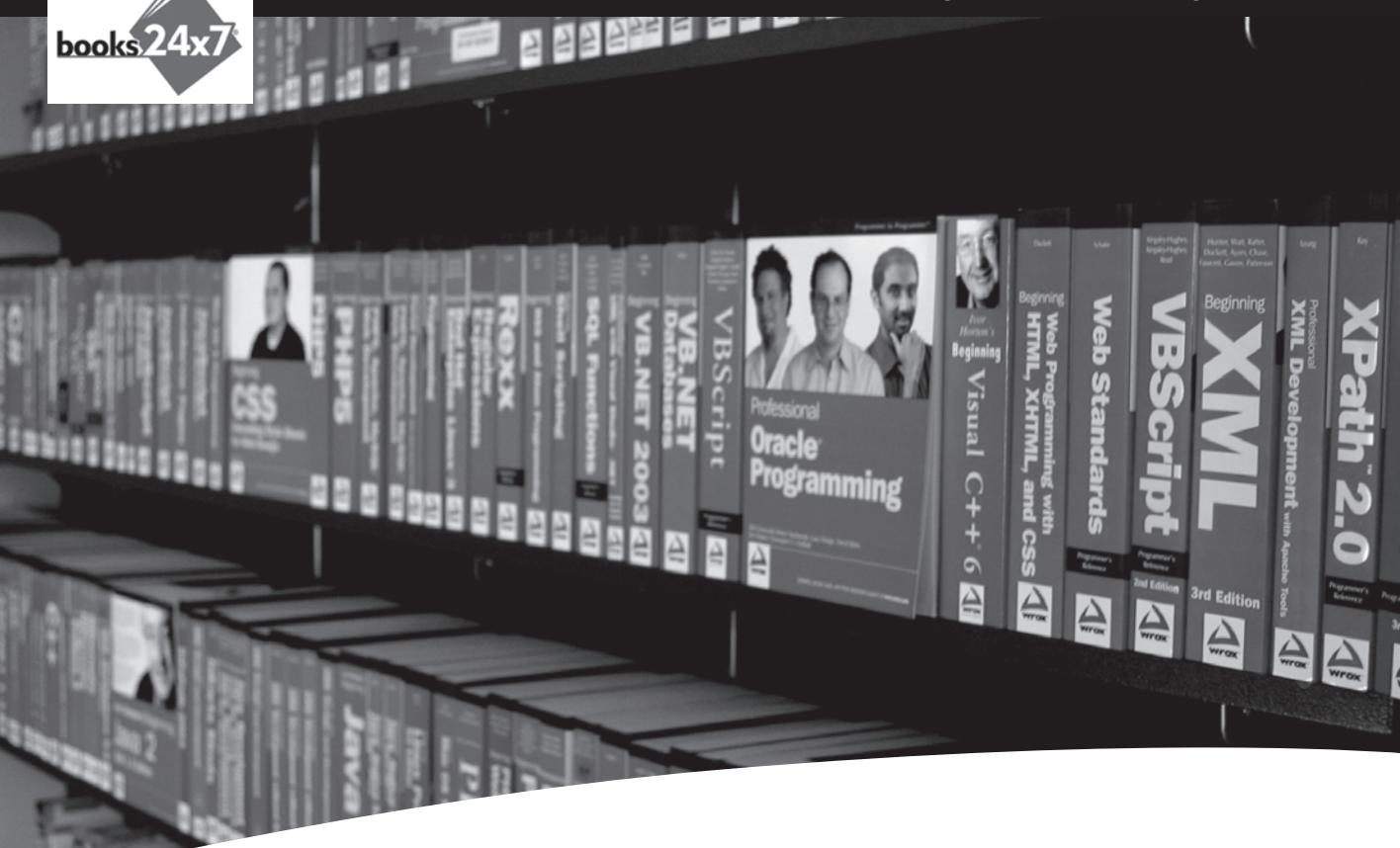
functionally constructing `XDocuments`,  
  300–302  
namespaces in C# LINQ to XML queries,  
  297–298  
removing expanded namespaces, 298–299

## **XML namespaces, in VB 9.0**

enabling IntelliSense for namespaces,  
  302–304  
multiple namespaces in XML literal queries,  
  304–305  
overview, 302

## **XSD, 48. See also LINQ to XSD**





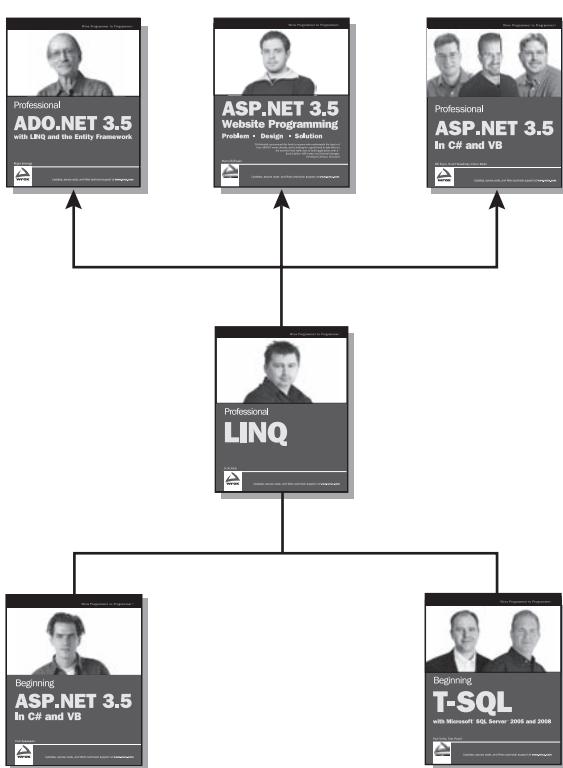
# Take your library wherever you go.

Now you can access more than 200 complete Wrox books online, wherever you happen to be! Every diagram, description, screen capture, and code sample is available with your subscription to the **Wrox Reference Library**. For answers when and where you need them, go to [wrox.books24x7.com](http://wrox.books24x7.com) and subscribe today!

## Find books on

- ASP.NET
- C#/C++
- Database
- General
- Java
- Mac
- Microsoft Office
- .NET
- Open Source
- PHP/MySQL
- SQL Server
- Visual Basic
- Web
- XML

# Professional **ADO.NET 3.5** with LINQ and the Entity Framework



## **Professional ADO.NET 3.5 with LINQ and the Entity Framework**

978-0-470-18261-1

This book is for intermediate to advanced developers of data-intensive .NET Windows and Web-based applications

## **ASP.NET 3.5 Website Programming Problem-Design-Solution**

978-0-470-18758-6

This book emphasizes n-tier ASP.NET Web application architectural design, something intermediate and advanced ASP.NET developers need and can't find anywhere else.

## **Professional ASP.NET 3.5: In C# and VB**

978-0-470-18757-9

This book is for programmers and developers who are looking to make the transition to ASP.NET 3.5 with Visual Studio 2008 and either C# 3.0 (2008) or Visual Basic 9 (2008).

## **Professional LINQ**

978-0-470-04181-9

This book is for developers who want to learn about LINQ and how it can benefit and enhance their applications.

## **Beginning ASP.NET 3.5**

978-0-470-18759-3

This book is for anyone who wants to build rich and interactive web sites that run on the Microsoft platform. No prior experience in web development is assumed.

## **Beginning T-SQL with Microsoft SQL Server 2005 and 2008**

978-0-470-25703-6

This book will provide you with an overview of SQL Server query operations and tools used with T-SQL, Microsoft's implementation of the SQL database query language.

**Enhance Your Knowledge  
Advance Your Career**

# Professional ADO.NET 3.5 with LINQ and the Entity Framework

LINQ and the Entity Framework are revolutionizing .NET database programming. With this book as your guide, you'll discover how to leverage these cutting-edge query and object/relational mapping technologies for enterprise-class computing. It provides you with hands-on coding techniques for data-intensive web and Windows projects. You'll also get quickly up to speed on LINQ technologies with the help of C# and VB programming examples.

Leading Microsoft database authority Roger Jennings first covers LINQ Standard Query Operators (SQOs) and domain-specific LINQ to SQL, LINQ to DataSet, and LINQ to XML implementations for querying generic collections. He then delves into the ADO.NET Entity Framework, Entity Data Model, Entity SQL (eSQL), and LINQ to Entities. Numerous code examples are integrated throughout the chapters that emulate real-world data sources and show you how to develop C# and VB web site/application or Windows projects.

The information in this book will give you the tools to create and maintain applications that are independent of the underlying relational data.

## What you will learn from this book

- A new approach to data access in ADO.NET 3.5 SP1
- Methods for working with advanced LINQ query operators and expressions
- Techniques for querying SQL Server® database with LINQ to SQL
- Approaches for integrating third-party and emerging LINQ implementations
- How to raise the level of data abstraction with the Entity Data Model
- Steps for creating design-time data sources from ObjectContext
- Ways to use the Entity Data Model as a data source

## Who this book is for

This book is for intermediate to advanced developers of data-intensive .NET web- and Windows-based applications.

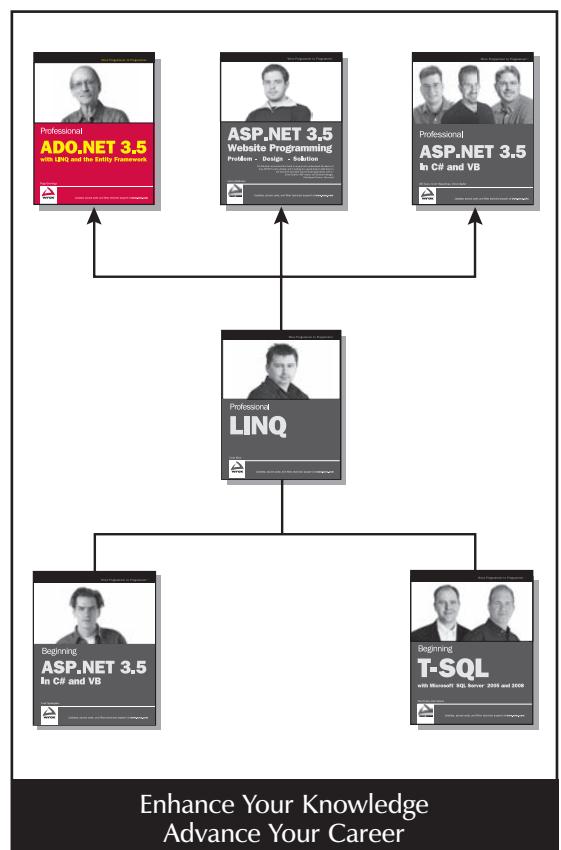
**Wrox Professional guides** are planned and written by working programmers to meet the real-world needs of programmers, developers, and IT professionals. Focused and relevant, they address the issues technology professionals face every day. They provide examples, practical solutions, and expert education in new technologies, all designed to help programmers do a better job.

p2p.wrox.com  
The programmer's resource center

Recommended Computer Book Categories	Programming
	Software Development

\$49.99 USA  
\$59.99 CAN

**Wrox™**  
An Imprint of  
 WILEY



Enhance Your Knowledge  
Advance Your Career

[www.wrox.com](http://www.wrox.com)

ISBN: 978-0-470-18261-1

