Microsoft

Developing Windows Azure and Web Services

Exam Ref 70-487

William Ryan
Wouter de Kort
Shane Milton

Exam Ref 70-487

Prepare for Microsoft Exam 70-487—and help demonstrate your real-world mastery of developing Windows Azure and web services. Designed for experienced developers ready to advance their status, *Exam Ref* focuses on the critical-thinking and decision-making acumen needed for success at the Microsoft Specialist level.

Focus on the expertise measured by these objectives:

- Accessing data
- Querying and manipulating data by using the Entity Framework
- Designing and implementing WCF Services
- Creating and consuming Web API-based services
- Deploying web applications and services

This Microsoft *Exam Ref:*

- Organizes its coverage by exam objectives.
- Features strategic, what-if scenarios to challenge you.
- Includes a 15% exam discount from Microsoft. Offer expires 12/31/2015. Details inside.

Developing Windows Azure and Web Services

About the Exam

Exam 70-487 is one of three Microsoft exams focused on the skills and knowledge necessary to create and deploy modern web applications and services.

About Microsoft Certification

Passing this exam earns you a **Microsoft Specialist** certification. You also receive credit toward a **Microsoft Certified Solutions Developer (MCSD)** certification that proves your ability to build innovative solutions across multiple technologies, both on-premises and in the cloud.

Exams 70-480, 70-486, and 70-487 are required for the MCSD: Web Applications certification.

See full details at:
microsoft.com/learning/certification

About the Authors

William Ryan is a solution architect who has served as a subject matter expert for several Microsoft exams, including Microsoft SQL Server and Windows Azure.

Wouter de Kort, MCSD, is an independent technical coach, trainer, and developer who works with C# and .NET, Entity Framework, and Team Foundation Server.

Shane Milton is a senior architect creating enterprise systems running in Windows Azure. He also designs cloud-based solutions for managing energy in homes and businesses.

microsoft.com/mspress

ISBN: 978-0-7356-7724-1

9 780735 677241 90000

U.S.A. $39.99
Canada $41.99
[Recommended]

Certification/Windows Server

Microsoft

Exam Ref 70-487: Developing Windows Azure and Web Services

William Ryan
Wouter de Kort
Shane Milton

Copyright © 2013 by Microsoft Press, Inc.

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

ISBN: 978-0-7356-7724-1

1 2 3 4 5 6 7 8 9 QG 8 7 6 5 4 3

Printed and bound in the United States of America.

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at *mspinput@microsoft.com*. Please tell us what you think of this book at *http://www.microsoft.com/learning/booksurvey*.

Microsoft and the trademarks listed at *http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx* are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

This book expresses the author’s views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation,
nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Acquisitions Editor: Jeff Riley

Developmental Editor: Ginny Bess Munroe

Production Editor: Kara Ebrahim

Editorial Production: Box Twelve Communications

Technical Reviewer: Shane Milton

Copyeditor: Nancy Sixsmith

Indexer: Angie Martin

Cover Design: Twist Creative • Seattle

Cover Composition: Ellie Volckhausen

Illustrator: Rebecca Demarest

Contents at a glance

Contents

What do you think of this book? We want to hear from you!

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

www.microsoft.com/learning/booksurvey/

What do you think of this book? We want to hear from you!
Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

www.microsoft.com/learning/booksurvey/

Introduction

Most books take a low-level approach, teaching you how to use individual classes and how to accomplish granular tasks. Like other Microsoft certification exams, this book takes a high-level approach, building on your knowledge of lower-level Microsoft Windows application development and extending it into application design. Both the exam and the book are so high level that there is little coding involved. In fact, most of the code samples in this book illustrate higher-level concepts.

The exam is written for developers who have three to five years of experience developing Web Services and at least one year of experience developing Web API and Azure solutions. Developers should also have at least three years of experience working with Relational Database Management systems and ADO.NET and at least one year of experience with the Entity Framework.

This book covers every exam objective, but it does not cover every exam question. Only the Microsoft exam team has access to the exam questions themselves, and Microsoft regularly adds new questions to the exam, making it impossible to cover specific questions. You should consider this book a supplement to your relevant real-world experience and other study materials. If you encounter a topic in this book that you do not feel completely comfortable with, use the links you'll find in the text to find more information and take the time to research and study the topic. Valuable information is available on MSDN, TechNet, and in blogs and forums.

Microsoft certifications

Microsoft certifications distinguish you by proving your command of a broad set of skills and experience with current Microsoft products and technologies. The exams and corresponding certifications are developed to validate your mastery of critical competencies as you design and develop, or implement and support, solutions with Microsoft products and technologies both on-premise and in the cloud. Certification brings a variety of benefits to the individual and to employers and organizations.

MORE INFO **ALL MICROSOFT CERTIFICATIONS**

For information about Microsoft certifications, including a full list of available certifications, go to *http://www.microsoft.com/learning/en/us/certification/cert-default.aspx*.

Acknowledgments

I'd like to thank Ginny Munroe and Shane Milton for the immense help they provided in preparing this book. My wife and daughter were extremely supportive throughout this stressful and difficult time. I'd also like to thank Walter Bellhaven and Herb Sewell for always keeping things uplifting.

Errata & book support

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site:

http://aka.ms/ER70-487/errata

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software is not offered through the addresses above.

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://www.microsoft.com/learning/booksurvey

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in touch

Let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*.

Preparing for the exam

Microsoft certification exams are a great way to build your resume and let the world know about your level of expertise. Certification exams validate your on-the-job experience and product knowledge. While there is no substitution for on-the-job experience, preparation through study and hands-on practice can help you prepare for the exam. We recommend that you round out your exam preparation plan by using a combination of available study materials and courses. For example, you might use the Exam Ref and another study guide for your "at home" preparation, and take a Microsoft Official Curriculum course for the classroom experience. Choose the combination that you think works best for you.

Note that this Exam Ref is based on publically available information about the exam and the author's experience. To safeguard the integrity of the exam, authors do not have access to the live exam.

CHAPTER 1

Accessing data

It's hard to find a modern software application that doesn't make extensive use of data access. Some exist, but particularly in the business realm, most have a heavy data access component. There are many ways to build data-centric applications and many technologies that can be used. Microsoft provides several, including ADO.NET, Entity Framework, and SQL Server. This objective covers about 24 percent of the exam's questions.

important

Have you read page xvii?

It contains valuable information regarding the skills you need to pass the exam.

Objectives in this chapter:

- Objective 1.1: Choose data access technologies
- Objective 1.2: Implement caching
- Objective 1.3: Implement transactions
- Objective 1.4: Implement data storage in Windows Azure
- Objective 1.5: Create and implement a WCF Data Services service
- Objective 1.6: Manipulate XML data structures

Objective 1.1: Choose data access technologies

There's no law that states that only one data access technology must be used per application. However, unless you have a specific need, it's generally advisable to pick a data access technology and stick with it throughout the application. Three obvious choices covered by this exam are *ADO.NET*, *Entity Framework (EF)*, and *WCF Data Services*.

This objective covers how to:

- Choose a technology (ADO.NET, Entity Framework, WCF Data Services) based on application requirements

Choosing a technology (ADO.NET, Entity Framework, WCF Data Services) based on application requirements

Choosing a data access technology is something that requires thought. For the majority of cases, anything you can do with one technology can be accomplished with the other technologies. However, the upfront effort can vary considerably. The downstream benefits and costs are generally more profound. WCF Data Services might be overkill for a simple one-user scenario. A console application that uses ADO.NET might prove much too limiting for any multiuser scenario. In any case, the decision of which technology to use should not be undertaken lightly.

Choosing ADO.NET as the data access technology

If tasked to do so, you could write a lengthy paper on the benefits of using ADO.NET as a primary data access technology. You could write an equally long paper on the downsides of using ADO.NET. Although it's the oldest of the technologies on the current stack, it still warrants serious consideration, and there's a lot to discuss because there's a tremendous amount of ADO.NET code in production, and people are still using it to build new applications.

ADO.NET was designed from the ground up with the understanding that it needs to be able to support large loads and to excel at security, scalability, flexibility, and dependability. These performance-oriented areas (security, scalability, and so on) are mostly taken care of by the fact that ADO.NET has a bias toward a *disconnected model* (as opposed to ADO's commonly used *connected model*). For example, when using individual commands such as INSERT, UPDATE, or DELETE statements, you simply open a connection to the database, execute the command, and then close the connection as quickly as possible. On the query side, you create a SELECT query, pull down the data that you need to work with, and immediately close the connection to the database after the query execution. From there, you'd work with a localized version of the database or subsection of data you were concerned about, make any changes to it that were needed, and then submit those changes back to the database (again by opening a connection, executing the command, and immediately closing the connection).

There are two primary reasons why a connected model versus disconnected model is important. First of all, connections are expensive for a relational database management system (RDBMS) to maintain. They consume processing and networking resources, and database systems can maintain only a finite number of active connections at once. Second, connections can hold locks on data, which can cause concurrency problems. Although it doesn't solve all your problems, keeping connections closed as much as possible and opening them only for short periods of time (the absolute least amount of time possible) will go a long way to mitigating many of your database-focused performance problems (at least the problems caused by the consuming application; database administrator (DBA) performance problems are an entirely different matter).

To improve efficiency, ADO.NET took it one step farther and added the concept of *connection pooling*. Because ADO.NET opens and closes connections at such a high rate, the minor overheads in establishing a connection and cleaning up a connection begin to affect performance. Connection pooling offers a solution to help combat this problem. Consider the scenario in which you have a web service that 10,000 people want to pull data from over the course of 1 minute. You might consider immediately creating 10,000 connections to the database server the moment the data was requested and pulling everybody's data all at the same time. This will likely cause the server to have a meltdown! The opposite end of the spectrum is to create one connection to the database and to make all 10,000 requests use that same connection, one at a time.

Connection pooling takes an in-between approach that works much better. It creates a few connections (let's say 50). It opens them up, negotiates with the RDBMS about how it will communicate with it, and then enables the requests to share these active connections, 50 at a time. So instead of taking up valuable resources performing the same nontrivial task 10,000 times, it does it only 50 times and then efficiently funnels all 10,000 requests through these 50 channels. This means each of these 50 connections would have to handle 200 requests in order to process all 10,000 requests within that minute. Following this math, this means that, if the requests can be processed on average in under ~300ms, you can meet this requirement. It can take ~100ms to open a new connection to a database. If you included that within that 300ms window, 33 percent of the work you have to perform in this time window is dedicated simply to opening and closing connections, and that will never do!

Finally, one more thing that connection pooling does is manage the number of active connections for you. You can specify the maximum number of connections in a connection string. With an ADO.NET 4.5 application accessing SQL Server 2012, this limit defaults to 100 simultaneous connections and can scale anywhere between that and 0 without you as a developer having to think about it.

ADO.NET compatibility

Another strength of ADO.NET is its cross-platform compatibility. It is compatible with much more than just SQL Server. At the heart of ADO.NET is the System.Data namespace. It contains many base classes that are used, irrespective of the RDBMS system. There are several vendor-specific libraries available (System.Data.SqlClient or System.Data.OracleClient, for instance) as well as more generic ones (System.Data.OleDb or System.Data.Odbc) that enable access to OleDb and Odbc-compliant systems without providing much vendor-specific feature access.

ADO.NET architecture

The following sections provide a quick overview of the ADO.NET architecture and then discuss the strengths and benefits of using it as a technology. A few things have always been and probably always will be true regarding database interaction. In order to do anything, you need to connect to the database. Once connected, you need to execute commands against the database. If you're manipulating the data in any way, you need something to hold the data that you just retrieved from the database. Other than those three constants, everything else can have substantial variability.

NOTE **PARAMETERIZE YOUR QUERIES**

There is no excuse for your company or any application you work on to be hacked by an injection attack (unless hackers somehow find a vulnerability in the DbParameter class that's been heretofore unknown). Serious damage to companies, individual careers, and unknowing customers has happened because some developer couldn't be bothered to clean up his dynamic SQL statement and replace it with parameterized code. Validate all input at every level you can, and at the same time, make sure to parameterize everything as much as possible. This one of the few serious bugs that is always 100 percent avoidable, and if you suffer from it, it's an entirely self-inflicted wound.

.NET Framework data providers

According to MSDN, .NET Framework *data providers* are described as "components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data." Table 1-1 lists the foundational objects of the data providers, the base class they derive from, some example implementations, and discussions about any relevant nuances.

TABLE 1-1 .NET Framework data provider overview

Provider object	Interface	Example items	Discussion
DbConnection	IDbConnection	SqlConnection, OracleConnection, EntityConnection, OdbcConnection, OleDbConnection	Necessary for any database interaction. Care should be taken to close connections as soon as possible after using them.
DbCommand	IDbCommand	SqlCommand, OracleCommand, EntityCommand, OdbcCommand, OleDbCommand	Necessary for all database interactions in addition to Connection. Parameterization should be done only through the Parameters collection. Concatenated strings should never be used for the body of the query or as alternatives to parameters.
DbDataReader	IDataReader	SqlDataReader, OracleDataReader, EntityDataReader, OdbcDataReader, OleDbDataReader	Ideally suited to scenarios in which speed is the most critical aspect because of its forward-only nature, similar to a Stream. This provides read-only access to the data.
DbDataAdapter	IDbDataAdapter	SqlDataAdapter, OracleDataAdapter, OdbcDataAdapter, OleDbDataAdapter	Used in conjunction with a Connection and Command object to populate a DataSet or an individual DataTable, and can also be used to make modifications back to the database. Changes can be batched so that updates avoid unnecessary roundtrips to the database.

Provider object	Interface	Example items	Discussion
DataSet	N/A	No provider-specific implementation	In-memory copy of the RDBMS or portion of RDBMS relevant to the application. This is a collection of DataTable objects, their relationships to one another, and other metadata about the database and commands to interact with it.
DataTable	N/A	No provider-specific implementation	Corresponds to a specific view of data, hether from a SELECT query or generated from .NET code. This is often analogous to a table in the RDBMS, although only partially populated. It tracks the state of data stored in it so, when data is modified, you can tell which records need to be saved back into the database.

The list in Table 1-1 is not a comprehensive list of the all the items in the System.Data (and provider-specific) namespace, but these items do represent the core foundation of ADO.NET. A visual representation is provided in Figure 1-1.

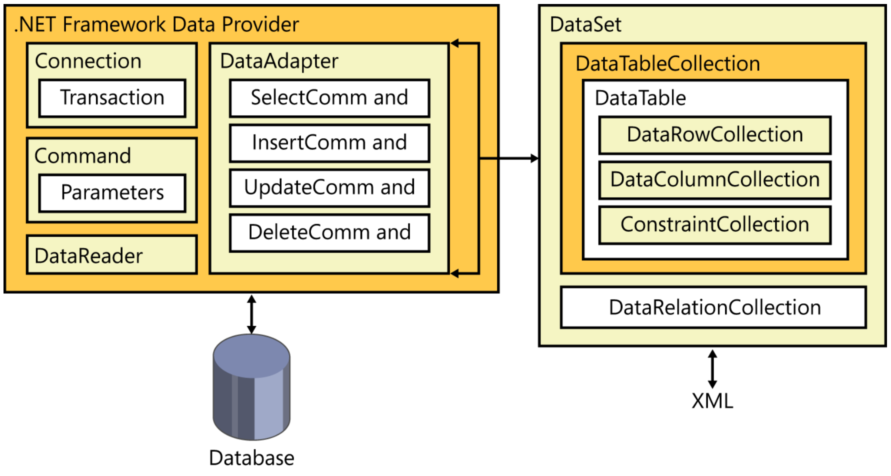


FIGURE 1-1 .NET Framework data provider relationships

DataSet or DataReader?

When querying data, there are two mechanisms you can use: a *DataReader* or a *DataAdapter*. These two options are more alike than you might think. This discussion focuses on the differences between using a DataReader and a DataAdapter, but if you said, “Every SELECT query operation you employ in ADO.NET uses a DataReader,” you’d be correct. In fact, when you use a DataAdapter and something goes wrong that results in an exception being thrown, you’ll typically see something like the following in the StackTrace of the exception: “System.InvalidOperationException: ExecuteReader requires an open and available Connection.” This exception is thrown after calling the Fill method of a SqlDataAdapter. Underneath the abstractions, a DataAdapter uses a DataReader to populate the returned DataSet or DataTable.

Using a DataReader produces faster results than using a DataAdapter to return the same data. Because the DataAdapter actually uses a DataReader to retrieve data, this should not surprise you. But there are many other reasons as well. Look, for example, at a typical piece of code that calls both:

```
[TestCase(3)]
public static void GetCustomersWithDataAdapter(int customerId)
{
    // ARRANGE
    DataSet customerData = new DataSet("CustomerData");
    DataTable customerTable = new DataTable("Customer");
    customerData.Tables.Add(customerTable);

    StringBuilder sql = new StringBuilder();
    sql.Append("SELECT FirstName, LastName, CustomerId, AccountId");
    sql.Append("  FROM [dbo].[Customer] WHERE CustomerId = @CustomerId ");

    // ACT
    // Assumes an app.config file has connectionString added to <connectionStrings>
section named "TestDB"
    using (SqlConnection mainConnection =
        new SqlConnection(ConfigurationManager.ConnectionStrings["TestDB"].
ConnectionString))
    {
       using (SqlCommand customerQuery = new SqlCommand(sql.ToString(), mainConnection))
        {
            customerQuery.Parameters.AddWithValue("@CustomerId", customerId);
            using (SqlDataAdapter customerAdapter = new SqlDataAdapter(customerQuery))
            {
                try
                {
                    customerAdapter.Fill(customerData, "Customer");
                }
                finally
                {
                    // This should already be closed even if we encounter an exception
                    // but making it explicit in code.
                    if (mainConnection.State != ConnectionState.Closed)
                    {
                        mainConnection.Close();
                    }
                }
            }
        }
    }

    // ASSERT
    Assert.That(customerTable.Rows.Count, Is.EqualTo(1), "We expected exactly 1 record
to be returned.");
    Assert.That(customerTable.Rows[0].ItemArray[customerTable.Columns["customerId"].
Ordinal],
        Is.EqualTo(customerId), "The record returned has an ID different than
expected.");
}
```

Query of Customer Table using SqlDataReader

```
[TestCase(3)]
public static void GetCustomersWithDataReader(int customerId)
{
    // ARRANGE
    // You should probably use a better data structure than a Tuple for managing your
data.
    List<Tuple<string, string, int, int>> results = new List<Tuple<string, string, int,
int>>();

    StringBuilder sql = new StringBuilder();
    sql.Append("SELECT FirstName, LastName, CustomerId, AccountId");
    sql.Append("  FROM [dbo].[Customer] WHERE CustomerId = @CustomerId ");

    // ACT
    // Assumes an app.config file has connectionString added to <connectionStrings>
section named "TestDB"
    using (SqlConnection mainConnection =
        new SqlConnection(ConfigurationManager.ConnectionStrings["TestDB"].
ConnectionString))
    {
        using (SqlCommand customerQuery = new SqlCommand(sql.ToString(),
mainConnection))
        {
            customerQuery.Parameters.AddWithValue("@CustomerId", customerId);
            mainConnection.Open();
            using (SqlDataReader reader = customerQuery.ExecuteReader(CommandBehavior.
CloseConnection))
            {
                try
                {
                    int firstNameIndex = reader.GetOrdinal("FirstName");
                    int lastNameIndex = reader.GetOrdinal("LastName");
                    int customerIdIndex = reader.GetOrdinal("CustomerId");
                    int accountIdIndex = reader.GetOrdinal("AccountId");

                    while (reader.Read())
                    {
                        results.Add(new Tuple<string, string, int, int>(
                          (string)reader[firstNameIndex], (string)reader[lastNameIndex],
                            (int)reader[customerIdIndex], (int)reader[accountIdIndex]));
                    }
                }
                finally
                {
                    // This will soon be closed even if we encounter an exception
                    // but making it explicit in code.
                    if (mainConnection.State != ConnectionState.Closed)
                    {
                        mainConnection.Close();
                    }
                }
            }
        }
    }

    // ASSERT
    Assert.That(results.Count, Is.EqualTo(1), "We expected exactly 1 record to be 
returned.");
    Assert.That(results[0].Item3, Is.EqualTo(customerId),
        "The record returned has an ID different than expected.");
}
```

Test the code and note the minimal differences. They aren't identical functionally, but they are close. The DataAdapter approach takes approximately 3 milliseconds (ms) to run; the DataReader approach takes approximately 2 ms to run. The point here isn't that the DataAdapter approach is 50 percent slower; it is approximately 1 ms slower. Any data access times measured in single-digit milliseconds is about as ideal as you can hope for in most circumstances. Something else you can do is use a profiling tool to monitor SQL Server (such as SQL Server Profiler) and you will notice that both approaches result in an identical query to the database.

IMPORTANT **MAKE SURE THAT YOU CLOSE EVERY CONNECTION YOU OPEN**

To take advantage of the benefits of ADO.NET, unnecessary connections to the database must be minimized. Countless hours, headaches, and much misery result when a developer takes a shortcut and doesn't close the connections. This should be treated as a Golden Rule: If you open it, close it. Any command you use in ADO.NET outside of a DataAdapter requires you to specifically open your connection. You must take explicit measures to make sure that it is closed. This can be done via a try/catch/finally or try/finally structure, in which the call to close the connection is included in the finally statement. You can also use the Using statement (which originally was available only in C#, but is now available in VB.NET), which ensures that the Dispose method is called on IDisposable objects. Even if you use a Using statement, an explicit call to Close is a good habit to get into. Also keep in mind that the call to Close should be put in the finally block, not the catch block, because the Finally block is the only one guaranteed to be executed according to Microsoft.

The following cases distinguish when you might choose a DataAdapter versus a DataReader:

- Although coding styles and technique can change the equation dramatically, as a general rule, using a DataReader results in faster access times than a DataAdapter does. (This point can't be emphasized enough: The actual code written can and will have a pronounced effect on overall performance.) Benefits in speed from a DataReader can easily be lost by inefficient or ineffective code used in the block.

- DataReaders provide multiple asynchronous methods that can be employed (BeginExecuteNonQuery, BeginExecuteReader, BeginExecuteXmlReader). DataAdapters on the other hand, essentially have only synchronous methods. With small-sized record sets, the differences in performance or advantages of using asynchronous methods are trivial. On large queries that take time, a DataReader, in conjunction with asynchronous methods, can greatly enhance the user experience.
- The Fill method of DataAdapter objects enables you to populate only DataSets and DataTables. If you're planning to use a custom business object, you have to first retrieve the DataSet or DataTables; then you need to write code to hydrate your business object collection. This can have an impact on application responsiveness as well as the memory your application uses.
- Although both types enable you to execute multiple queries and retrieve multiple return sets, only the DataSet lets you closely mimic the behavior of a relational database (for instance, add Relationships between tables using the Relations property or ensure that certain data integrity rules are adhered to via the EnforceConstraints property).
- The Fill method of the DataAdapter completes only when all the data has been retrieved and added to the DataSet or DataTable. This enables you to immediately determine the number of records in any given table. By contrast, a DataReader can indicate whether data was returned (via the HasRows property), but the only way to know the exact record count returned from a DataReader is to iterate through it and count it out specifically.
- You can iterate through a DataReader only once and can iterate through it only in a forward-only fashion. You can iterate through a DataTable any number of times in any manner you see fit.
- DataSets can be loaded directly from XML documents and can be persisted to XML natively. They are consequently inherently serializable, which affords many features not natively available to DataReaders (for instance, you can easily store a DataSet or a DataTable in Session or View State, but you can't do the same with a DataReader). You can also easily pass a DataSet or DataTable in between tiers because it is already serializable, but you can't do the same with a DataReader. However, a DataSet is also an expensive object with a large memory footprint. Despite the ease in doing so, it is generally ill-advised to store it in Session or Viewstate variables, or pass it across multiple application tiers because of the expensive nature of the object. If you serialize a DataSet, proceed with caution!
- After a DataSet or DataTable is populated and returned to the consuming code, no other interaction with the database is necessary unless or until you decide to send the localized changes back to the database. As previously mentioned, you can think of the dataset as an in-memory copy of the relevant portion of the database.

IMPORTANT **FEEDBACK AND ASYNCHRONOUS METHODS**

Using any of the asynchronous methods available with the SqlDataReader, you can provide feedback (although somewhat limited) to the client application. This enables you to write the application in such a way that the end user can see instantaneous feedback that something is happening, particularly with large result sets. DataReaders have a property called HasRows, which indicates whether data was returned from the query, but there is no way to know the exact number of rows without iterating through the DataReader and counting them. By contrast, the DataAdapter immediately makes the returned record count for each table available upon completion.

EXAM TIP

Although ADO.NET is versatile, powerful, and easy to use, it's the simplest of the choices available. When studying for the exam, you won't have to focus on learning every nuance of every minor member of the System.Data or System.Data.SqlClient namespace. This is a Technical Specialist exam, so it serves to verify that you are familiar with the technology and can implement solutions using it. Although this particular objective is titled "Choose data access technology," you should focus on how you'd accomplish any given task and what benefits each of the items brings to the table. Trying to memorize every item in each namespace is certainly one way to approach this section, but focusing instead on "How do I populate a DataSet containing two related tables using a DataAdapter?" would probably be a much more fruitful endeavor.

Why choose ADO.NET?

So what are the reasons that would influence one to use traditional ADO.NET as a data access technology? What does the exam expect you to know about this choice? You need to be able to identify what makes one technology more appropriate than another in a given setting. You also need to understand how each technology works.

The first reason to choose ADO.NET is consistency. ADO.NET has been around much longer than other options available. Unless it's a relatively new application or an older application that has been updated to use one of the newer alternatives, ADO.NET is already being used to interact with the database.

The next reason is related to the first: stability both in terms of the evolution and quality of the technology. ADO.NET is firmly established and is unlikely to change in any way other than feature additions. Although there have been many enhancements and feature improvements, if you know how to use ADO.NET in version 1.0 of the .NET Framework, you will know how to use ADO.NET in each version up through version 4.5. Because it's been around so long, most bugs and kinks have been fixed.

ADO.NET, although powerful, is an easy library to learn and understand. Once you understand it conceptually, there's not much left that's unknown or not addressed. Because it has been around so long, there are providers for almost every well-known database, and many lesser-known database vendors have providers available for ADO.NET. There are examples showing how to handle just about any challenge, problem, or issue you would ever run into with ADO.NET.

One last thing to mention is that, even though Windows Azure and cloud storage were not on the list of considerations back when ADO.NET was first designed, you can use ADO.NET against Windows Azure's SQL databases with essentially no difference in coding. In fact, you are encouraged to make the earlier SqlDataAdapter or SqlDataReader tests work against a Windows Azure SQL database by modifying only the connection string and nothing else!

Choosing EF as the data access technology

EF provides the means for a developer to focus on application code, not the underlying "plumbing" code necessary to communicate with a database efficiently and securely.

The origins of EF

Several years ago, Microsoft introduced *Language Integrated Query (LINQ)* into the .NET Framework. LINQ has many benefits, one of which is that it created a new way for .NET developers to interact with data. Several flavors of LINQ were introduced. *LINQ-to-SQL* was one of them. At that time (and it's still largely the case), RDBMS systems and *object oriented programming (OOP)* were the predominant metaphors in the programming community. They were both popular and the primary techniques taught in most computer science curriculums. They had many advantages. OOP provided an intuitive and straightforward way to model real-world problems.

The relational approach for data storage had similar benefits. It has been used since at least the 1970s, and many major vendors provided implementations of this methodology. Most all the popular implementations used an ANSI standard language known as *Structured Query Language (SQL)* that was easy to learn. If you learned it for one database, you could use that knowledge with almost every other well-known implementation out there. SQL was quite powerful, but it lacked many useful constructs (such as loops), so the major vendors typically provided their own flavor in addition to basic support for ANSI SQL. In the case of Microsoft, it was named *Transact SQL* or, as it's commonly known, *T-SQL*.

Although the relational model was powerful and geared for many tasks, there were some areas that it didn't handle well. In most nontrivial applications, developers would find there was a significant gap between the object models they came up with via OOP and the ideal structures they came up with for data storage. This problem is commonly referred to as impedance mismatch, and it initially resulted in a significant amount of required code to deal with it. To help solve this problem, a technique known as *object-relational mapping (ORM, O/RM, or O/R Mapping*) was created. LINQ-to-SQL was one of the first major Microsoft initiatives to build an ORM tool. By that time, there were several other popular ORM tools, some open source and some from private vendors. They all centered on solving the same essential problem.

Compared to the ORM tools of the time, many developers felt LINQ-to-SQL was not powerful and didn't provide the functionality they truly desired. At the same time that LINQ-to-SQL was introduced, Microsoft embarked upon the EF initiative. EF received significant criticism early in its life, but it has matured tremendously over the past few years. Right now, it is powerful and easy to use. At this point, it's also widely accepted as fact that the future of data access with Microsoft is the EF and its approach to solving problems.

The primary benefit of using EF is that it enables developers to manipulate data as domain-specific objects without regard to the underlying structure of the data store. Microsoft has made (and continues to make) a significant investment in the EF, and it's hard to imagine any scenario in the future that doesn't take significant advantage of it.

From a developer's point of view, EF enables developers to work with entities (such as Customers, Accounts, Widgets, or whatever else they are modeling). In EF parlance, this is known as the conceptual model. EF is responsible for mapping these entities and their corresponding properties to the underlying data source.

To understand EF (and what's needed for the exam), you need to know that there are three parts to the EF modeling. Your .NET code works with the conceptual model. You also need to have some notion of the underlying storage mechanism (which, by the way, can change without necessarily affecting the conceptual model). Finally, you should understand how EF handles the mapping between the two.

EF modeling

For the exam and for practical use, it's critical that you understand the three parts of the EF model and what role they play. Because there are only three of them, that's not difficult to accomplish.

The conceptual model is handled via what's known as the *conceptual schema definition language (CSDL)*. In older versions of EF, it existed in a file with a .csdl extension. The data storage aspect is handled through the *store schema definition language (SSDL)*. In older versions of EF, it existed in a file with an .ssdl file extension. The mapping between the CSDL and SSDL is handled via the *mapping specification language (MSL)*. In older versions of EF, it existed in a file with an .msl file extension. In modern versions of EF, the CSDL, MSL, and SSDL all exist in a file with an .edmx file extension. However, even though all three are in a single file, it is important to understand the differences between the three.

Developers should be most concerned with the conceptual model (as they should be); database folk are more concerned with the storage model. It's hard enough to build solid object models without having to know the details and nuances of a given database implementation, which is what DBAs are paid to do. One last thing to mention is that the back-end components can be completely changed without affecting the conceptual model by allowing the changes to be absorbed by the MSL's mapping logic.

Compare this with ADO.NET, discussed in the previous section. If you took any of the samples provided and had to change them to use an Oracle database, there would be major changes necessary to all the code written. In the EF, you'd simply focus on the business objects and let the storage model and mappings handle the change to how the data came from and got back to the database.

Building EF models

The early days of EF were not friendly to the technology. Many people were critical of the lack of tooling provided and the inability to use industry-standard architectural patterns because they were impossible to use with EF. Beginning with version 4.0 (oddly, 4.0 was the second version of EF), Microsoft took these problems seriously. By now, those complaints have been addressed.

There are two basic ways you can use the set of *Entity Data Model (EDM)* tools to create your conceptual model. The first way, called *Database First*, is to build a database (or use an existing one) and then create the conceptual model from it. You can then use these tools to manipulate your conceptual model. You can also work in the opposite direction in a process called *Model First*, building your conceptual model first and then letting the tools build out a database for you. In either case, if you have to make changes to the source or you want to change the source all together, the tools enable you to do this easily.

NOTE **CODE FIRST**

An alternative way to use EF is via a *Code First* technique. This technique enables a developer to create simple classes that represent entities and, when pointing EF to these classes, enables the developer to create a simple data tier that just works. Although you are encouraged to further investigate this technique that uses no .edmx file, the exam does not require that you know how to work with this technique much beyond the fact that it exists. As such, anywhere in this book that discusses EF, you can assume a Model First or Database First approach.

When you create a new EF project, you create an .edmx file. It's possible to create a project solely from XML files you write yourself, but that would take forever, defeat the purpose for using the EF, and generally be a bad idea. The current toolset includes four primary items that you need to understand:

- The *Entity Model Designer* is the item that creates the .edmx file and enables you to manipulate almost every aspect of the model (create, update, or delete entities), manipulate associations, manipulate and update mappings, and add or modify inheritance relationships.
- The *Entity Data Model Wizard* is the true starting point of building your conceptual model. It enables you to use an existing data store instance.

- The *Create Database Wizard* enables you to do the exact opposite of the previous item. Instead of starting with a database, it enables you to fully build and manipulate your conceptual model, and it takes care of building the actual database based on the conceptual model.
- The *Update Model Wizard* is the last of the tools, and it does exactly what you'd expect it to. After your model is built, it enables you to fully modify every aspect of the conceptual model. It can let you do the same for both the storage model and the mappings that are defined between them.

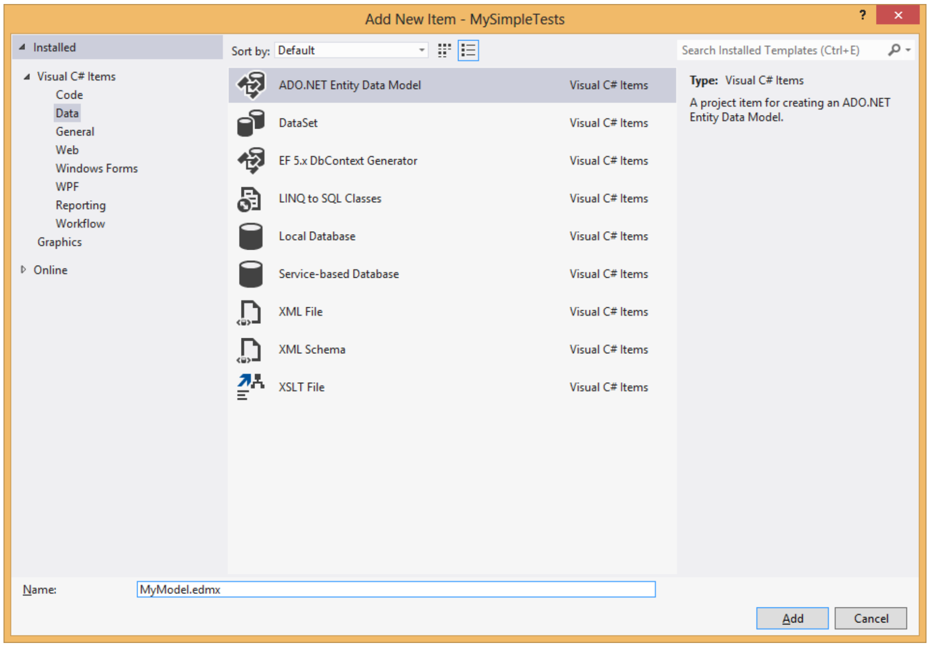
There's one other tool that's worth mentioning, although it's generally not what developers use to interact with the EF. It's known as the *EDM Generator* and is a command-line utility that was one of the first items built when the EF was being developed. Like the combination of the wizard-based tools, it enables you to generate a conceptual model, validate a model after it is built, generate the actual C# or VB.NET classes that are based off of the conceptual model, and also create the code file that contains model views. Although it can't hurt to know the details of how this tool works, the important aspects for the exam focus on each of the primary components that go into an EDM, so it is important to understand what each of those are and what they do.

Building an EF Model using the Entity Data Model Wizard

This section shows you how to use the tools to build a simple model against the TestDB created in the beginning of Chapter 1. You can alternatively manually create your models and use those models to generate your database if you alter step 3 and choose Empty Model instead.

However, before you begin, make sure that your TestDB is ready to go, and you're familiar with how to connect to it. One way is to ensure that the tests back in the ADO.NET section pass. Another way is to ensure that you can successfully connect via *SQL Server Management Studio (SSMS)*. For the included screen shots, the EF model is added to the existing MySimpleTests project.

1. First, right-click on your Project in Solution Explorer and add a New Item.
2. In the Add New Item dialog box, select Visual C# Items → Data on the left and ADO.NET Entity Data Model in the middle (don't let the name of this file type throw you off because it does include "ADO.NET" in the name). Name this **MyModel.edmx** and click Add (see Figure 1-2).


Add New Item - MySimpleTests

Installed

Visual C# Items

Code

Data

General

Web

Windows Forms

WPF

Reporting

Workflow

Graphics

Online

Sort by: Default

Search Installed Templates (Ctrl+E)

ADO.NET Entity Data Model	Visual C# Items
DataSet	Visual C# Items
EF 5.x DbContext Generator	Visual C# Items
LINQ to SQL Classes	Visual C# Items
Local Database	Visual C# Items
Service-based Database	Visual C# Items
XML File	Visual C# Items
XML Schema	Visual C# Items
XSLT File	Visual C# Items

Type: Visual C# Items

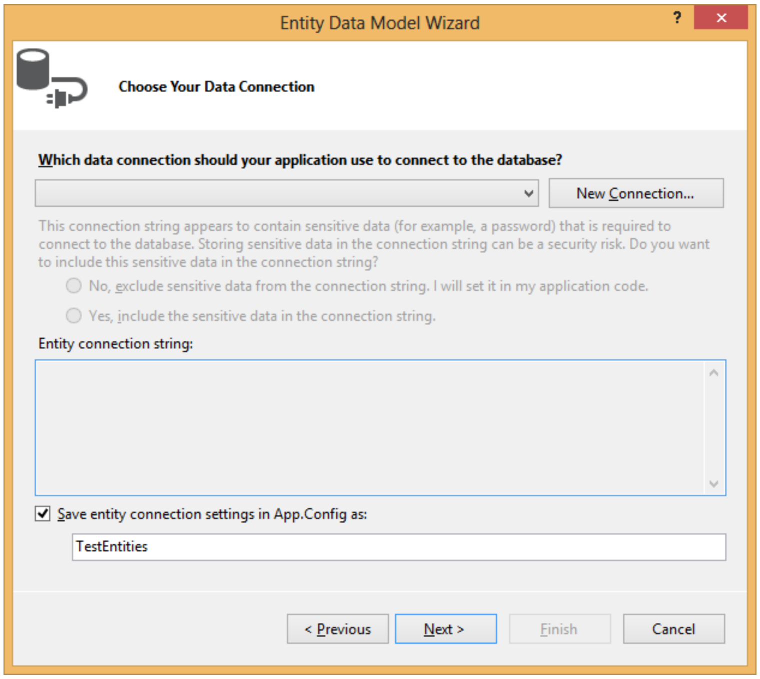
A project item for creating an ADO.NET Entity Data Model.

Name: MyModel.edmx

FIGURE 1-2 ADO.NET Entity Data Model Wizard dialog box

3. In the Entity Data Model Wizard dialog box, select Generate From Database and click Next.

4. Next, the Entity Data Model Wizard requires that you connect to your database. Use the New Connection button to connect and test your connection. After you're back to the Entity Data Model Wizard dialog box, ensure that the check box to save your connection settings is selected and name it **TestEntities** (see Figure 1-3).



Entity Data Model Wizard

Choose Your Data Connection

Which data connection should your application use to connect to the database?

This connection string appears to contain sensitive data (for example, a password) that is required to connect to the database. Storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

No, exclude sensitive data from the connection string. I will set it in my application code.

Yes, include the sensitive data in the connection string.

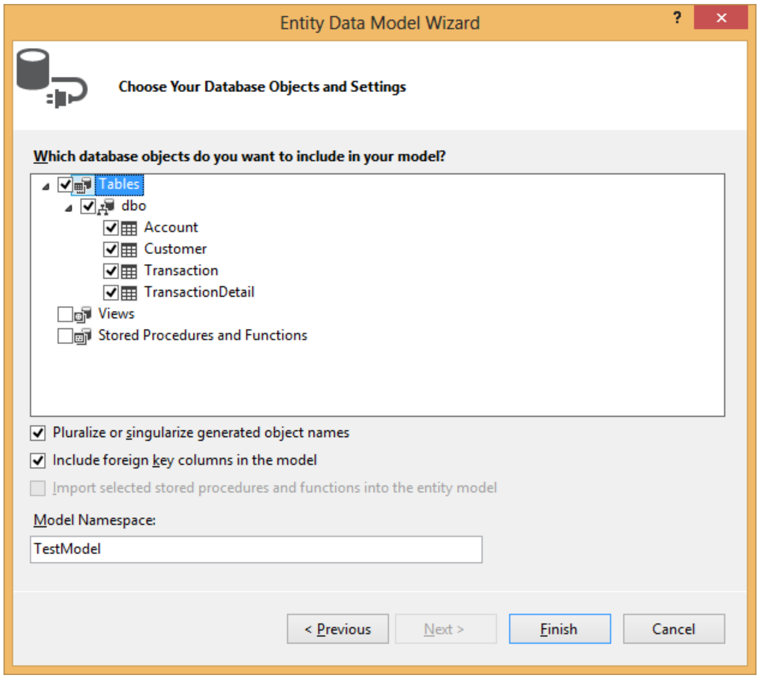
Entity connection string:

[✓] Save entity connection settings in App.Config as:

TestEntities

FIGURE 1-3 Choose Your Data Connection dialog box

5. In the next screen of the Entity Data Model Wizard, select all the tables and select both check boxes under the database objects. Finally, for the namespace, call it **TestModel** and click Finish (see Figure 1-4).


Entity Data Model Wizard

Choose Your Database Objects and Settings

Which database objects do you want to include in your model?

- [✓] Tables
  - [✓] dbo
    - [✓] Account
    - [✓] Customer
    - [✓] Transaction
    - [✓] TransactionDetail
- [ ] Views
- [ ] Stored Procedures and Functions

[✓] Pluralize or singularize generated object names

[✓] Include foreign key columns in the model

[ ] Import selected stored procedures and functions into the entity model

Model Namespace:

TestModel

FIGURE 1-4 Choose Your Database Objects And Settings dialog box

6. You should now see the Entity Model Designer in a view that looks similar to an entity relationship diagram shown in Figure 1-5.

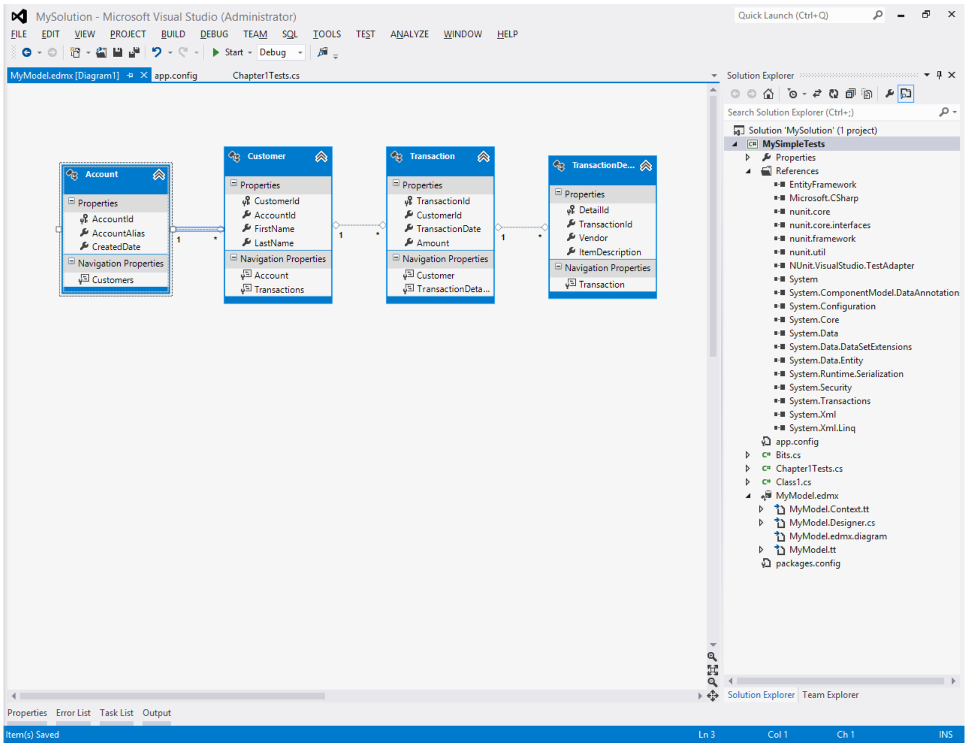


FIGURE 1-5 Entity Model view

After generating your EF models, you now have a fully functioning data tier for simple consumption! Quickly test this to investigate everything you just created. See the code for a quick test of your new EF data tier:

Entity Framework test

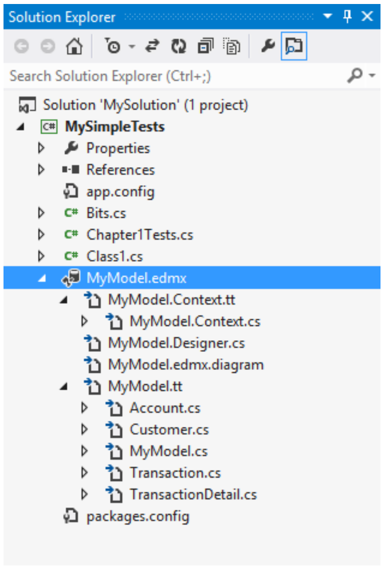
```
[TestCase(3)]
public static void GetCustomerById(int customerId)
{
    // ARRANGE
    TestEntities database = new TestEntities();

    // ACT
    Customer result = database.Customers.SingleOrDefault(cust => cust.CustomerId ==
customerId);

    // ASSERT
    Assert.That(result, Is.Not.Null, "Expected a value. Null here indicates no record
with this ID.");
    Assert.That(result.CustomerId, Is.EqualTo(customerId), "Uh oh!");
}
```

There are a few things to note about what happens in this test. First, the complexity of the code to consume EF is completely different compared with the prior ADO.NET tests! Second, this test runs in approximately 4 ms compared with the 2–3 ms for ADO.NET. The difference here isn't so much 50–100 percent, but rather 1–2 ms for such a simple query. Finally, the query that runs against SQL Server in this case is substantially different from what was run with the ADO.NET queries for two reasons: EF does some ugly (although optimized) aliasing of columns, tables, and parameters; and this performs a SELECT TOP (2) to enforce the constraint from the use of the Linq SingleOrDefault command.

MORE INFO **SINGLE VERSUS FIRST IN LINQ**

When using LINQ, if you have a collection and you want to return a single item from it, you have two obvious options if you ignore nulls (four if you want to handle nulls). The First function effectively says, "Give me the first one of the collection." The Single function, however, says, "Give me the single item that is in the collection and throw an exception if there are more or fewer than one item." In both cases, the xxxOrDefault handles the case when the collection is empty by returning a null value. A bad programming habit that many developers have is to overuse the First function when the Single function is the appropriate choice. In the previous test, Single is the appropriate function to use because you don't want to allow for the possibility of more than one Customer with the same ID; if that happens, something bad is going on!

As shown in Figure 1-6, there's a lot behind this .edmx file you created. There are two things of critical importance and two things that are mostly ignorable. For now, ignore the MyModel.Designer.cs file, which is not currently used, and ignore the MyModel.edmx.diagram file, which is used only for saving the layout of tables in the visual designer.



```
Solution 'MySolution' (1 project)
  MySimpleTests
    Properties
    References
    app.config
    Bits.cs
    Chapter1Tests.cs
    Class1.cs
    MyModel.edmx
      MyModel.Context.tt
        MyModel.Context.cs
      MyModel.Designer.cs
      MyModel.edmx.diagram
      MyModel.tt
        Account.cs
        Customer.cs
        MyModel.cs
        Transaction.cs
        TransactionDetail.cs
    packages.config
```

FIGURE 1-6 EF-generated files

MORE INFO **T4 CODE GENERATION**

T4 text template files can often be identified by the .tt extension. *T4* is the templating and code generation engine that EF uses to generate code so you don't have to manage it yourself. It's useful if you know how to automatically generate code in bulk based on known variables. In this case, your known variables are your database structure and, more importantly, your conceptual model. Because you created the various models in your .edmx file, you now have a handy definition file for your conceptual model.

First, look at the MyModel.tt file and then its Customer.cs file. Double-click the MyModel.tt file and you'll see the contents of this T4 text template. This template generates simple classes that represent the records in your tables in your database. Now open the Customer.cs file. The only two pieces of this file that might be new to you are the ICollection and HashSet types. These are simple collection types that are unrelated to EF, databases, or anything else. ICollection<T> is a simple interface that represents a collection; nothing special here. A HashSet<T> is similar to a generic List<T>, but is optimized for fast lookups (via hashtables, as the name implies) at the cost of losing order.

T4-Generated Customer.cs

```
public partial class Customer
{
    public Customer()
    {
        this.Transactions = new HashSet<Transaction>();
    }

    public int CustomerId { get; set; }
    public int AccountId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }

    public virtual Account Account { get; set; }
    public virtual ICollection<Transaction> Transactions { get; set; }
}
```

Next, look at the file MyModel.Context.tt generated: MyModel.Context.cs. There are three important things to see in this file. First, the TestEntities class inherits the DbContext class. This class can be thought of as the EF runtime that does all the magic work. The DbContext API was introduced with EF 4.1, and Microsoft has an excellent series of documentation on how to work with this API at *http://msdn.microsoft.com/en-us/data/gg192989.aspx*.

Inheriting DbContext enables it to talk to a database when coupled with the .edmx file and the connection string in the config file. Looking at this example, notice that the parameter passes to the base constructor. This means that it depends on the config file having a properly configured EF connection string named TestEntities in the config file. Take a look at it and notice how this connection string differs from the one you used for the ADO.NET tests. Also notice the DbSet collections. These collections are what enable us to easily work with each table in the database as if it were simply a .NET collection. Chapter 2, "Querying and manipulating data by using the Entity Framework," investigates this in more detail.

T4-Generated MyModel.Context.cs

```
public partial class TestEntities : DbContext
{
    public TestEntities()
        : base("name=TestEntities")
    {
    }

    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
        throw new UnintentionalCodeFirstException();
    }

    public DbSet<Account> Accounts { get; set; }
    public DbSet<Customer> Customers { get; set; }
    public DbSet<Transaction> Transactions { get; set; }
    public DbSet<TransactionDetail> TransactionDetails { get; set; }
}
```

Entity Data Model Designer

When you double-click the .edmx file and are given the ERD-like view, you invoke the *EDM Designer*. This is the primary tool for manipulating your models, whether you're manually creating a new one, updating your existing models based on changes in the schema of your data, or choosing to manually tweak your conceptual model without changes to your storage model.

When you have the Designer open, there is a window that is useful to have at times that can be opened by clicking View → Other Windows → Entity Data Model Browser. This is especially useful as your .edmx begins to cover a large number of tables and you have difficulty locating a particular one.

If you want to manage your models in the Designer, typically you'd just right-click anywhere inside the design canvas of the .edmx model file you're working with or on a specific entity or property that you want to manage. If you want to update your models based on data schema changes, it can be done with the Update Model Wizard by right-clicking in the design canvas and selecting Update Model From Database. In the Update Model Wizard, you can choose to add new tables or simply click Finish for it to pull in schema changes to existing tables.

The options that might raise some questions are likely to be the Inheritance and Complex Type options. Otherwise, editing the conceptual model is straightforward. Inheritance is a way to create entities with OOP class hierarchies that is primarily used in Code First and Model First (but not Database First) scenarios. With Inheritance, you can have both an Employee and

Customer type that inherits a Person type. Although an Employee might have a Salary field, a Customer certainly wouldn't. Behind-the-scenes in your data store, this can be represented in a couple different ways.

The default mapping strategy, called *Table per Hierarchy (TPH)*, is the most straightforward way to do this. It creates a single table for all objects in an inheritance hierarchy, and it simply has nullable columns for fields that aren't common across all types. It also adds a Discriminator column so EF can keep track of the type of each individual record. This strategy is often the best balance of tradeoffs available because it provides the best performance. The primary disadvantage is that your data is slightly denormalized.

The prevailing alternative strategy, called *Table per Type (TPT)*, creates a table for your base type that has all common fields in it, a table for each child type that stores the additional fields, as well as an ID to the base type's record that stores the common fields. The multiple inheriting types' tables are linked to one another via a foreign key that has a shared primary key value. In this case, to get all the data for a single entity, EF must perform a join across multiple tables. The primary advantage is that your data is properly normalized. The primary disadvantage is that your performance suffers.

Your other mapping options include *Table per Concrete Type (TPC)* and *Mixed Inheritance*. These two options are not currently supported in the EDM Designer, although the EF runtime does support them. The exam will likely not cover these mapping options, nor are they usually your most practical choice to use because of the limited tooling support for them.

A *Complex Type* is a logical designation for a common group of fields on multiple entities. When you have a Complex Type, your conceptual model has an object to work with for easier usability of these repeated groups of fields (an example is a date range). Many tables in many databases have a StartDate field and a StopDate field. You could take these two fields, identify them as being parts of a Complex Type called a DateRange, and then your conceptual model could specify when an entity has a Date Range instead of arbitrary date fields.

The only other major item is the actual editing of the conceptual entity items. If you select any given entity on the conceptual model canvas, right-click it, and select Add New, you'll see a menu with the following items:

- Scalar Property
- Navigation Property
- Complex Property
- Association
- Inheritance
- Function Import

From there, the remaining features are self-explanatory. Following is a brief description of each:

- **Table Mapping** enables you to change the relationship to the underlying table. For instance, you can add fields, remove fields, or change fields from the specific property originally mapped to.
- **Stored Procedure Mapping** is based on the Stored Procedures defined in the database. It enables you specify an Insert, Update, and Delete function. Select queries are already handled by the language semantics of LINQ.
- **Update Model from Database** is the same behavior described previously in the canvas-based designer.
- **Generate Database from Model** is the same behavior described previously in the canvas-based designer.
- **Validate** is the same behavior described previously in the canvas-based designer that ensures the validity of the .edmx file.

Using the generated items

So far, everything has involved the .edmx files and their respective components. It's critical to understand each of the pieces and why they matter when designing an application, but understanding how to generate a model is only one part of the equation. A tremendous amount of time is spent developing and maintaining business objects and their interaction with the database, so having this handled for you is beneficial. But just as important is to understand what you do with everything after it is generated.

In the simplest scenario, the conceptual model provides you with classes that map directly to the underlying tables in your RDBMS structure (unless, of course, you specify otherwise). These classes are actual C# or VB.NET code, and you can work with them just like any other class you create. To understand things and prepare for the exam, I highly recommend that you generate a few different .edmx files (try both Database First and Model First) and examine them closely.

At the top level, the primary object you'll work with is an ObjectContext or DbContext object. Choose a name for the context early on in the wizard. It can be changed just like everything else, but the important takeaway for the exam is that this is the primary item from which everything else flows. If you do want to change the name, however, you simply have focus in the Designer for the .edmx file and set the Entity Container Name value in the Properties window.

ObjectContext versus DbContext

Older versions of EF did not have DbContext. Instead, ObjectContext was the equivalent class that funneled most of the functionality to the developer. Modern versions of EF still support the ObjectContext object, and you can even consume modern EF in much the same way as the older versions. Suffice it to say, if you're beginning a new project, DbContext is where you should look to for your EF needs. However, both flavors of the API are important from a real-world legacy code perspective as well as a testing perspective for this exam. In other words, be prepared to answer questions about both ways of using EF. Fortunately, many of the concepts from the one flavor are directly applicable to the other flavor.

The steps for creating an .edmx are intended for creating a DbContext usage scenario. If you want to work in an ObjectContext scenario, you have three primary options. The intimate details of these approaches are beyond this book; you are encouraged to investigate these approaches:

- **Legacy approach** Follow similar steps using Visual Studio 2008 or 2010 to create an .edmx file and its corresponding ObjectContext and entities.
- **Downgrading your entities** Take the .edmx file that was generated, open the Properties window, and set focus in the Designer's canvas. From here, change the Code Generation Strategy from None to Default in the Properties window. Finally, delete the two .tt files listed as children to your .edmx file in the Solution Explorer window.
- **Hybrid approach** Get the ObjectContext (via a nonobvious approach) from the DbContext and work with the ObjectContext directly. Note that even with modern versions of EF, in some rare and advanced scenarios this is still required:

Hybrid approach

```
public static ObjectContext ConvertContext(DbContext db)
{
    return ((IObjectContextAdapter)db).ObjectContext;
}

[TestCase(3)]
public static void GetCustomerByIdOnObjectContext(int customerId)
{
    // ARRANGE
    TestEntities database = new TestEntities();
    ObjectContext context = ConvertContext(database);

    // ACT
    ObjectSet<Customer> customers = context.CreateObjectSet<Customer>("Customers");
    Customer result = customers.SingleOrDefault(cust => cust.CustomerId == customerId);
    //Customer result = database.Customers.SingleOrDefault(cust => cust.CustomerId ==
customerId);

    // ASSERT
    Assert.That(result, Is.Not.Null, "Expected a value. Null here indicates no record
with this ID.");
    Assert.That(result.CustomerId, Is.EqualTo(customerId), "Uh oh!");
}
```

As you can tell from this test with the commented line of code, basic consumption of a DbContext is quite similar to that of an ObjectContext. In fact, most T4 templated instances of ObjectContexts already have properties pregenerated for you, so you can simply access the equivalent ObjectSet<Customer> collection directly off of the generated class that derives ObjectContext. This simple consumption truly is identical to that of a the generated DbContext-derived class and its DbSet collection.

ObjectContext management

Two important things happen when an ObjectContext constructor is called. The generated context inherits several items from the ObjectContext base class, including a property known as ContextOptions and an event named OnContextCreated. The ContextOptions class has five primary properties you can manipulate:

- LazyLoadingEnabled
- ProxyCreationEnabled
- UseConsistentNullReferenceBehavior
- UseCSharpNullComparisonBehavior
- UseLegacyPreserveChangesBehavior

The LazyLoadingEnabled property is set when any instance of the context is instantiated and is used to enable lazy loading. If left unspecified, the default setting is true. Lazy loading enables entities to be loaded on-demand without thought by the developer. Although this can be handy in some scenarios, this behavior can have very serious performance implications depending on how relationships are set up. A good deal of thought should be taken into consideration regarding this enabling or disabling lazy loading. Feature development and even application architecture might change one way or another based on the use of this feature. And some architectures make it necessary to disable this feature.

In EF, lazy loading is triggered based on a Navigation Property being accessed. By simply referencing a navigation property, that entity is then loaded. As an example, let's take the Customer class and the related Transaction class. If LazyLoadingEnabled is set to true, and you load one customer, you'll just get back data just for that customer record. But if you later access a Navigation Property within the same ObjectContext, another roundtrip to the database will be made to fetch that data. If you loop through a collection of entities (in this case, the Transactions), an individual roundtrip is made to the database for each entity. If you have LazyLoadingEnabled set to false, you have two options. You can either use explicit lazy loading with the ObjectContext's LoadProperty() method or you can use eager loading with the ObjectSet's Include() method. With explicit lazy loading, you must write code to explicitly perform this additional roundtrip to the database to conditionally load the data. With eager loading, you must specify up front what related data you want loaded. Although explicit lazy loading, like regular lazy loading, can reduce the amount of data flowing back and forth, it's a chatty pattern, whereas eager loading is a chunky pattern.

NOTE **CHATTY VERSUS CHUNKY**

Assume it takes 100 ms per MB of data. In just transfer times alone, that puts you at 2.5 seconds versus 1 second transferring data. However, assume there is also a 100 ms delay for every round trip (quite common on the Internet and is not too terribly uncommon in high-availability environments). In addition to data transfer times, you must also add 100 ms to the 25 MB transfer, but a huge 10 seconds to the 10 MB transfer times. So in the end, it would take only 2.6 seconds for 25 MB, but 11 seconds for 10 MB, simply because you chose to be chatty instead of chunky with the architectural decision! Now if the transfer times were the same but the latency was only 1 ms (back to the two VMs on one host), you would be looking at 2.501 seconds versus 1.1 seconds, and the chattier of the two options would have been the better choice!

Of the five ContextOption properties, LazyLoadingEnabled is by far the most important one with the most serious implications for your application. Although it is inappropriate to refer to the other properties as unimportant, lazy loading can very quickly make or break your application.

The ProxyCreationEnabled property is one of the other properties of the ObjectContextOptions class that simply determines whether proxy objects should be created for custom data classes that are persistence-ignorant, such as *plain old common object (POCO)* entities (POCO entities are covered more in Chapter 2). It defaults to true and, unless you have some specific reason to set it to false, it generally isn't something you need to be overly concerned about.

This property is available only in newer versions of EF. A very common problem with this property occurs if you initially target .NET Framework 4.5 with your application code, reference this property, and then later drop down to .NET 4.0 where this property does not exist.

This functionality that this property controls is a little confusing if you disable lazy loading and load a Customer object from the TestDB database. If you inspect the AccountId, you can see a value of the parent record. But if you inspect the Account navigation property, it is null because it has not yet been loaded. Now with the UseConsistentNullReferenceBehavior property set to false, you can then attempt to set the Account navigation property to null, and nothing will happen when you try to save it. However, if this setting is set to true, the act of setting the Account navigation property to null attempts to sever the relationship between the two records upon saving (which results in an error because the AccountId is not a nullable field). If, on the other hand, you had the Customer object loaded and its Account property was also loaded (either by eager loading or lazy loading), the UseConsistentNullReferenceBehavior setting has no effect on setting the Account property to null.

As the name implies, C# NullComparison behavior is enabled when set to true; otherwise, it is not enabled. The main implication of this property is that it changes queries that require null comparisons. For example, take a look at a test and see what happens. To set the data up for this test, ensure that your Account table has three records in it: one with an AccountAlias set to Home, one with an AccountAlias set to Work, and one with AccountAlias set to null. You can tweak the second parameter of the test to other numbers if you want to work with a different number of records:

Null comparison behavior

```
[TestCase(true, 2)]
[TestCase(false, 1)]
public static void GetAccountsByAliasName(bool useCSharpNullBehavior, int recordsToFind)
{
    // ARRANGE
    TestEntities database = new TestEntities();
    ObjectContext context = ConvertContext(database);
    ObjectSet<Account> accounts = context.CreateObjectSet<Account>("Accounts");

    // ACT
    context.ContextOptions.UseCSharpNullComparisonBehavior = useCSharpNullBehavior;
    int result = accounts.Count(acc => acc.AccountAlias != "Home");

    // ASSERT
    Assert.That(result, Is.EqualTo(recordsToFind), "Uh oh!");
}
```

Behind the scenes, when this setting is enabled, EF automatically tweaks your query to also include cases in which AccountAlias is set to null. In other words, depending on whether it is enabled or disabled, you might see the two following queries hitting your database:

```
SELECT COUNT(*) FROM Account WHERE AccountAlias <> 'Home' – Disabled
SELECT COUNT(*) FROM Account WHERE AccountAlias <> 'Home' OR AccountAlias IS NULL --
Enabled
```

This property can be set to true or false and is quite confusing. This setting causes EF to use either the .NET Framework 4.0 (and newer) functionality for MergeOption.PreserveChanges or use the .NET Framework 3.5 SP1 version when merging two entities on a single context. Setting this to true gives you some control over how to resolve Optimistic Concurrency Exceptions while attaching an entity to the ObjectContext when another copy of that same entity already exists on the context. Starting with.NET Framework version 4.0, when you attach an entity to the context, the EF compares the current local values of unmodified properties against those already loaded in the context. If they are the same, nothing happens. If they are different, the state of the property is set to Modified. The legacy option changes this behavior. When the entity is attached, the values of unmodified properties are not copied over; instead, the context's version of those properties is kept. This might be desirable, depending on how you want to handle concurrency issues in your application. Dealing with concurrency issues and all the implications that come with it is well beyond the scope of both the exam and this book.

ObjectContext entities

Understanding the structure of ObjectContext is critical for the exam. In addition, you need to understand the structure of the Entity classes that are generated for it. Thanks to the appropriate T4 templates, each entity that you define in the conceptual model gets a class generated to accompany it. There are a few options for the types of entities that can be generated. In EF 4.0, the default base class for entities was EntityObject. Two other types of entities could also be created: POCO entities (no required base class for these) and Self-Tracking entities (STEs) (no base class, but they implemented IObjectWithChangeTracker and INotifyPropertyChanged). Chapter 2 covers POCO entities and STE in more detail, but very little knowledge of them is required for the exam.

The ObjectContext entities of significant interest inherit from the EntityObject class. The class declaration for any given entity resembles the following:

```
public partial class Customer : EntityObject
{
    /* ... */
}
```

EXAM TIP

As noted earlier, you need to pay attention to the derived class that the given context is derived from because it is likely to be included in the exam. The same is the case with entities that derive from the EntityObject base class. These entities have several attributes decorating the class definition that you should be familiar with. There are also several similar distinctions surrounding properties of each entity class (or *attributes*, as they are called in the conceptual model).

Next, each entity will be decorated with three specific attributes:

- EdmEntityTypeAttribute
- SerializableAttribute
- DataContractAttribute

So the full declaration looks something like the following:

```
[EdmEntityTypeAttribute(NamespaceName = "MyNameSpace", Name = "Customer")]
[Serializable()]
[DataContractAttribute(IsReference = true)]
public partial class Customer : EntityObject
{
    /* ... */
}
```

The Entity class needs to be serializable. If you want to persist the object locally, pass it between tiers, or do much else with it, it has to be serializable. Because ObjectContext's incarnations of the EF were created to work in conjunction with WCF and WCF Data Services, the

DataContractAttribute must decorate the class (more on this later in the chapter, but for now, just understand that this is required for WCF's default serialization).

The last attribute is the EdmEntityTypeAttribute, which contains the item's namespace and the Name. Remember that the power of the EF comes from the fact that it gives you such tremendous designer support. In order to do that, you'd need either a lot of code or a lot of metadata, and in most cases, a decent amount of both. This EdmEntityTypeAttribute does little more than tell the world that the class is an EDM type and then provides a basic amount of supporting information about it.

Look at the definition of each class that's generated corresponding to each entity and as much as possible in the .edmx. Each property needs to be marked with the DataMember attribute (more about this in Objective 1.5). In addition to the DataMember attribute, the EdmScalarProperty Attribute decorates each property definition. The two main properties that it defines are EntityKeyProperty and IsNullable. The EntityKeyProperty simply indicates whether it is an EntityKey or not. IsNullable simply indicates whether this value allows Nulls.

C# 3.0 introduced a new kind of property: *auto-properties*. An example of an auto-property might look like this:

```
public int CustomerID { get; set; }
```

No matching private variables were declared, and the backing variables were just inferred. They were still there, but it was essentially syntactical sugar to save a few lines of code. The one big "gotcha" was that it was usable only when the get and set accessors did nothing but return or accept a value. If any logic happened inside of the accessor, you still needed to define a backing variable. An example of an EntityKey property on an entity follows:

```
[EdmScalarPropertyAttribute(EntityKeyProperty=true, IsNullable=false)]
[DataMemberAttribute()]
public global::System.Int32 CustomerId
{
    get
    {
        return _CustomerId;
    }
    set
    {
        if (_CustomerId != value)
        {
            OnCustomerIdChanging(value);
            ReportPropertyChanging("CustomerId");
            _CustomerId = StructuralObject.SetValidValue(value, "CustomerId");
            ReportPropertyChanged("CustomerId");
            OnCustomerIdChanged();
        }
    }
}
private global::System.Int32 _CustomerId;
partial void OnCustomerIdChanging(global::System.Int32 value);
partial void OnCustomerIdChanged();
```

The get accessor does nothing but return the private backing variable. The set accessor verifies that the value you're attempting to set is different from the value it currently holds. This makes sense, if you're only going to set something back to itself; why waste the effort? Once it verifies that the value has in fact changed, the OnChanged event is raised passing in the new value. Next, the ReportPropertyChanging event is raised, passing in a string corresponding to the entity name. The ReportPropertyChanging and ReportPropertyChanged events (the latter of which is called shortly afterward) are both members of the EntityObject base class and are used for fundamentally different purposes than the entity-specific events are.

So after the ReportPropertyChanging event is raised, the backing variable is updated with the new value. After that's been successfully completed, the ReportPropertyChanged event is raised, indicating that the operation has completed as far as the UI or host is concerned. Finally, the OnXChangedEvent specific to that entity is raised and processing is finished.

NOTE **THOSE WEREN'T EVENTS**

Even though these last few paragraphs just called some things "events," what you see in the generated code aren't events. Technically, they're nothing more than partial methods that are invoked. Even if some other portion of this partial class implements them, doing so doesn't make them events. However, the underlying implementation of these partial functions does, in fact, raise the events we're discussing. So although you might not actually see any generated code working with events here, this generated code does result in the discussed events being raised.

Why choose the EF?

The EF is just one of many choices you have available for data access. Compared with ADO.NET data services, you could argue that it's more similar than different; however, the opposite is true. Microsoft has made a tremendous investment in the EF and continues to, which is indicative of its future plans and where it intends data access to go. The EF provides many benefits over previous data access technologies, and now that the toolset has matured, there's not much downside. Following are the benefits:

- There is tremendous tooling support that enables you to build and maintain the data access elements of your application in a much shorter time than ADO.NET would.
- You can focus on dealing with business objects of your application and don't have to spend your time overly concerned about the underlying data store.
- It allows for a clear separation of concerns between the conceptual model and the underlying data store.

- The data store can be changed (quite dramatically) without having to rewrite core application logic. Microsoft has made a tremendous investment into the EF, and all indications point to the fact it has made a firm commitment to the success of the EF.
- The EF is built to fit right in with WCF, which makes them very complementary technologies.

Choosing WCF Data Services as the data access technology

The development landscape has changed over the past few years, even faster and more dynamically than it has in other equivalent periods. Three of the highly notable changes are these:

- *Open Data Protocol (OData)*
- *Representational State Transfer (REST)*
- *JavaScript Object Notation (JSON)*

WCF Data Services, originally called ADO.NET Data Services, is the main mechanism of Microsoft to deal with exposing these features to developers. WCF Data Services are particularly well-suited to applications that are exposed via *Service-Oriented Architecture (SOA)* and enable you to build applications that outsiders (and insiders for that matter) can easily consume and work with. *Web Services and Windows Communication Foundation (WCF)* were certainly big steps forward in this regard, but WCF Data Services takes it all one step farther in making the whole process a much easier endeavor.

Changing the nature of data access

If you think of how you typically approach getting a set of data, the first thing that probably comes to mind is writing a SQL query. If you've been a user of EF, you might think of an Entity Model. If you want to make the application accessible to outsiders, you will likely consider building and publishing a web service.

Consumers of that web service would have to go through several steps to build an application that consumed your service. This typically involves using the *Web Service Definition Language (WSDL)* to let the consumer know what methods were available. From there, they'd query it and likely use a tool to automate referencing the service. In the .NET world, the toolset would look at the metadata and build a proxy class that handled all the network communication. The proxy class would typically be written in C# or VB.NET and, once generated, the developer would use it just like any other class.

OData

OData is an abbreviated form of the name Open Data Protocol. OData.org describes OData in the following way:

> *The Open Data Protocol (OData) is a Web protocol for querying and updating data that provides a way to unlock your data and free it from silos that exist in applications today. OData does that by applying and building upon Web Technologies such as HTTP, Atom Publishing Protocol (AtomPub) and JSON to provide access to information from a variety of applications, services and stores. The protocol emerged from experiences implementing AtomPub clients and servers in a variety of products over the past several years. OData is being used to expose and access information from a variety of sources including, but not limited to, relational database, file systems, content management systems and traditional websites.*
>
> *OData is consistent with the way the Web works—it makes a deep commitment to URIs for resource identification and commits to an HTTP-based, uniform interface for interacting with those resources (just like the Web). This commitment to core Web principles allows OData to enable a new level of data integration and interoperability across a broad range of clients, servers, services, and tools.*

JSON

According to json.org, *JSON* is described as follows:

> *JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate... These properties make JSON an ideal data-interchange language.*

JSON is easy to use and consume. It is built on two fundamental mechanisms:

- **A name/value pair of data** You have a name that you reference it by and a value that corresponds to that name.
- **An ordered list of values** The list can take just about any form and serves as a container or collection to hold the name/value pairs.

Because JSON is little more than text, there are no proprietary file formats to deal with, it is straightforward, and it lends itself well to the mechanisms that comprise WCF Data Services.

WCF Data Services as data access mechanisms

Because Microsoft moved its original Web Services implementation (*.asmx*) to WCF, service names now end in the .svc extension. WCF is powerful and provides features well beyond what will be discussed here, but for the time being, it is worth it to understand that, underlying it all, the name was originally ADO.NET Data Services and has since been changed to WCF Data Services. Despite this change in name, the underlying technology is still much the same in that it intends to be a data access mechanism.

In a typical scenario, you build a WCF Data Service by performing the following steps:

1. Create an ASP.NET Web Application.
2. Use the EF to define an EDM.
3. Add a Data Service to the Web Application (which is there just to host the application and expose it over the web).
4. Enable access to the service.

Why choose WCF Data Services?

WCF Data Services has advantages and disadvantages. Because it uses the EF as a foundation, most (but not all) scenarios that were appropriate for one would be appropriate for the other. WCF Data Services would be overkill for simple one-user scenarios. On the other hand, it provides several out-of-the-box benefits that ADO.NET and the EF do not. These include the following:

- Because data is exposed, when using OData, resources are addressable via URIs. That alone makes it very nonproprietary and opens the service up to be consumed by everyone from Microsoft developers, to folks writing apps for iPhones, and everyone in between.
- WCF Data Services are accessed over HTTP, and almost everyone is familiar with HTTP these days and has access to it. Someone can literally query your application and get data back without having to write a single line of code.
- OData feeds can take several different forms, including Atom, JSON, or XML. All those formats can be represented as text, so many problems with firewalls, security, installing applications, and so forth immediately disappear.
- Very powerful queries can be constructed with very simple semantics. Without knowing SQL, someone would have a hard time returning all the Customers from your in-house database, let alone returning a customer with the last name of Doe.

In SQL, they'd have to have a specific toolset, know the connection information to your database, and then write something like "SELECT * FROM Customers WHERE LastName = 'Doe' (and yes, in a production app the = 'Doe' should be parameterized, but this is just for the sake of illustration). To do the same with WCF Data Services, all they'd need to do is enter *http://my company uri/MyApplication/MyService.svc/Customers* in the address bar of Internet Explorer to get a list of all the customers. To return the exact equivalent of what was shown before, they'd simply need to make one small change: *http://my company uri/MyApplication/MyService.svc/Customers('Doe').*

To take full advantage of WCF Data Services, the EF is meant to be used. But the EF can get its data from just about anywhere, and the tooling support makes it very easy to swap out or modify underlying data sources.

WCF Data Services provide a feature known as *Interceptors*. Interceptors enable you to build in quite sophisticated business logic, as you'll see when you build a WCF Data Service.

Thought experiment

Dealing with legacy issues

In the following thought experiment, apply what you learned about the "Choosing a data access technology" objective to determine the data access strategy for new application development at your company. You can find answers to these questions in the "Answers" section at the end of this chapter.

Contoso has several applications that currently use ADO.NET to interact with the database. Each application used to run locally inside the corporate network. Now, there's tremendous pressure to expose many of them to the outside world. You're tasked with deciding whether ADO.NET should continue to be used or whether a move should be made to another technology. If a move away from ADO.NET is made, you need to decide which other technology should be used.

With this in mind, answer the following questions:

1. **Should you continue building applications with ADO.NET?**
2. **Does using the EF provide any benefit toward the stated goal that ADO.NET doesn't?**
3. **Would WCF Data Services provide any benefit toward the stated goal that ADO.NET doesn't?**

Objective summary

1. ADO.NET has been around the longest and has several advantages. It does not require persistent connections to the underlying data store, enables you to access virtually all major database implementations, and enables you to access data through custom objects or through objects specifically suited to the tasks at hand (DataSet and DataTable).
2. By using the EF, developers can focus on the conceptual model (solving business logic and dealing with business issues) without being overly concerned with the underlying data store. EF is specifically focused on working with an entity, but not quite as much as working with bulk data all at once.
3. With EF, the underlying data stores can be easily modified and changed without requiring changes of the client code. The EF enables you to generate models based on an existing database schema, or it can work in reverse and build a database schema based on a conceptual model. Both are supported and quite easy to learn and use.
4. WCF Data Services let your applications provide universal data access. The consumption of WCF Data Services is not tied to any proprietary technology, so can be consumed by both Microsoft and non-Microsoft technologies. WCF Data Services are meant to be used in conjunction with the EF on the back end. They provide a very fast and easy way to build applications and make the data easily accessible to any consumer.

Objective review

Answer the following questions to test your knowledge of the information in this objective. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter

1. You are building an ADO.NET EF application. You need to add a property of type NVARCHAR(50) named Notes to the underlying database table named Customer and have it reflected in the Customer entity. What should you do?
   - **A.** Drop the Customer table and re-create it with the new field; then choose the Generate Database From Model option.
   - **B.** Add the field to the underlying table, add a Scalar property of type String to the Customer entity, and update the mapping.
   - **C.** Run the Update Model Wizard.
   - **D.** Run the EDM Generator.

2. You have been asked to choose a data access technology to retrieve data from a SQL Server database and have it exposed over the web. The application will be consumed exclusively by external parties that have no access to the internal database. Which data access technologies should you recommend? (Choose all that apply.)
   - **A.** LINQ-to-SQL
   - **B.** ADO.NET Entity Framework
   - **C.** WCF Data Services
   - **D.** ADO.NET Data Services
3. You are working with an EDM and suspect there's a problem with the mappings. Which file should you look at to see the underlying code?
   - **A.** CSDL file
   - **B.** SSDL file
   - **C.** MSL file
   - **D.** EDMX file

Objective 1.2: Implement caching

The way applications are used and the corresponding demands have changed in just the last few years. Users are much more demanding, less tolerant of inconveniences, and much more concerned about performance. It is commonly mentioned that if users have to wait more than five seconds for a page to load, they will typically leave the page.

Just try to imagine how many users an application like Bing has, and imagine what would happen if Bing kept open connections to the underlying databases for each user. There are several techniques that have been developed to deal with such issues, but one of the most effective ones is caching. Some data (for instance, stock quotes) needs to be the absolute latest every time it is accessed in some applications. Other data, such as a list of states that populates a combo box control, seldom changes and needs to be accessed rarely after initially retrieved.

If used effectively, data caching can greatly enhance responsiveness and provide a much better user experience. However, it must be used carefully. To see why, simply imagine a hospital's patient management system. During intake, the patient is unconscious, but the staff can get much of what it needs from information in the patient's wallet. They enter the data and assume that it is immediately cached and updated only every 12 hours. Imagine then that a member of the patient's family arrives and informs the staff that the patient is a Type 1 diabetic and has several medical allergies. If this information isn't immediately reflected in the application, the patient could end up facing severe consequences because doctors made decisions based on stale information.

This objective covers how to:

- Understand caching options
- Use the ObjectCache
- Use the HttpContext.Cache

Understanding caching options

There are several caching mechanisms you need to be familiar with for the exam. The first is the ObjectCache class. Although there are several different ways to cache data via the .NET Framework, not to mention your own or custom approaches, for the exam you should focus on the ObjectCache and the HttpContext.Cache items in particular. ASP.NET (which is not the specific target of this exam, but has aspects that might be covered) has Application State, Session State, and View State, all of which can be considered caching mechanisms. Outside of specific mechanisms, items such as DataSets, DataTables, or serializable business objects can all be serialized and therefore be considered cached in such a state. Finally, classes and properties (and methods, for that matter) can all be defined as static, which can have the effective result of providing cache functionality. For this portion of the exam, you should focus on the ObjectCache and its features, as well as HttpContext.Cache. Although many different items and technologies might qualify semantically as a cache, this chapter looks at those features built specifically to address caching needs.

NOTE **WINDOWS AZURE–PROVIDED CACHING OPTIONS**

Although this version of the exam does not require you know the caching options available specifically in Windows Azure, you certainly should be aware of them. These options will certainly be on future versions of the exam! You currently have two primary options: *Shared or Co-Located Caching* and *Dedicated Caching*.

With Co-Located Caching, Windows Azure takes a sliver of your Web Role or Worker Role instance's resources (memory being one of the most important resources) and uses that to store the cached data. The value here is that you don't have to pay for anything extra in terms of price; it is just a slight performance hit. If your cache needs to store only a few MB of data, this is a very good option.

With Dedicated Caching, you add a new role type to your project, a Cache Worker Role, which results in an instance whose entire purpose in life is to manage and host the cache. The benefits are that you can easily manage the number of these instances independent of other instances, and hosting very large amounts of cached data doesn't directly affect performance on your other servers or your application.

Using the ObjectCache

To put data into the ObjectCache, it first needs to be created. The current constructor is protected, so it isn't instantiated using the new keyword. Instead, you use the MemoryCache. Default property to get an instance.

Conveniently, the ObjectCache class alone provides nearly all the functionality for caching. Outside of the Extension methods, it is simple to memorize all its properties (there are four). Depending on the version of the .NET Framework being used, there are about 10 methods, and most of them have names used extensively throughout the Framework, so they are dead giveaways as to what they do. When you exclude methods that have unquestionably clear features (for example, GetEnumerator and GetCount), you're left with Add, AddOrGetExisting, Get, and Set. Some classes, particularly those that inherit from powerful base classes, have large numbers of extension methods, or implement multiple interfaces, and won't be so easy to memorize by just looking at a few code snippets, but many classes will. So when preparing for this exam, look at the following code samples to get a visual understanding of how things work and what the most commonly used major features are.

Very simple ObjectCache example

```
[TestCase("Cache1", 1)]
[TestCase("Cache1", 2)]
[TestCase("Cache1", 3)]
public void CanCache(string key, int value)
{
    // ARRANGE
    ObjectCache cache = MemoryCache.Default;
    var policy = new CacheItemPolicy
    {
        AbsoluteExpiration = new DateTimeOffset(DateTime.Now.AddMinutes(1))
    };

    // ACT
    cache.Remove(key);
    cache.Add(key, value, policy);
    int fetchedValue = (int)cache.Get(key);

    // ASSERT
    Assert.That(fetchedValue, Is.EqualTo(value), "Uh oh!");
}
```

This code example has various numbers in the cache for up to one minute. Notice that it first removes the item from cache. If the item isn't in cache, this gracefully fails and continues; you can never be guaranteed that an item is in a cache at any moment in time, even if you just checked for it 3 ms ago. You remove it because the second time this test runs, the item from the first run will still be in there. The way the Add method works in this case is that it calls the underlying AddOrGetExisting method (more on this later), so it doesn't actually replace an existing value when used this way. Try the tests again, but delete the line calling Remove and see what happens.

After you create an instance of the cache and put items in it, you can verify the existence of those items by using the default indexer on the ObjectCache and the string value you chose as a key whenever you added the item. Assume, for instance, that you added a string value that was the PublicKeyValue of an encryption key. The following code tells you whether the item was in the ObjectCache (using the as keyword enables you to attempt the reference without throwing an exception if the item is not present). You don't have to use this technique specifically, but you will want to verify that something exists in the cache before trying to reference it just like as would with any other collection.

Safely checking cached items

```
String encryptionPublicKey = CacheInstance["encryptionpublickey"] as String;
if (String.IsNullOrWhiteSpace(encryptionPublicKey))
{
    // Build encryption policy here and cache it.
}
```

There are several mechanisms for adding items into the CacheInstance:

- Add(CacheItem, CacheItemPolicy)
- Add(String key, Object value, DateTimeOffset absoluteExpiration)
- Add(String key, Object value, CacheItemPolicy policy, String regionName)
- AddOrGetExisting(CacheItem, CacheItemPolicy)
- AddOrGetExisting(String key, Object value, DateTimeOffset absoluteExpiration)
- AddOrGetExisting(String key, Object value, CacheItemPolicy policy, String regionName)
- Set(CacheItem, CacheItemPolicy)
- Set(String key, Object value, CacheItemPolicy policy, String regionName)

As noted, the items in the cache are stored as key/value pairs. When examining the overloads for the Add and AddOrGetExisting methods, overloads support both entering in a key/value pair or a CacheItem. The three primary properties in the CacheItem base class are, unsurprisingly, Key, Value, and RegionName. If you examine each of the overloads for Add or AddOrGetExisting, you'll note that they're functionally equivalent in the sense that you can use a Key to reference the item, or you can use a CacheItem to reference it, which in turn, provides the Key, and, if desired, the Value and RegionName. The cache implementation is allowed to store the cached data in any internal format it wants. The only restrictions are that the APIs provide that the data is represented in CacheItem objects, not that those CacheItem objects are actually stored internally.

It's worth noting that many of these are functional equivalents. For instance, according to the MSDN documentation, the Add method that uses a CacheItem and CacheItemPolicy as parameters does the following: When overridden in a derived class, it tries to insert a cache entry into the cache as a CacheItem instance and adds details about how the entry should be evicted. It describes the equivalent Set method as follows: When overridden in a derived class, it inserts the cache entry into the cache as a CacheItem instance, specifying information about how the entry will be evicted.

The main differences between the two reside in the return types. With the Add method, a value of True is returned if no item with that key was present and the insertion was successful. When a value of False is returned, there is no need to add it because it already exists, so insertion of the value actually failed. Internally, the Add method actually calls the AddOrGet Existing method.

The Set method has no return value, and if the key is not present, it inserts the value. If the key is present, it is updated with the value from the CacheItem parameter. In the previous test case, Set would have been a more appropriate method to use, but would have missed showing this very important point. You should review these on MSDN if they aren't immediately intuitive, but they each work the same way. The only real nuance is setting the CacheItem-Policy, which is also quite straightforward.

CacheItemPolicy

Using a CacheItemPolicy is simple. Once you instantiate an instance of it, you'll then want to set an expiration policy by using either an AbsoluteExpiration or a SlidingExpiration. The difference between the two is apparent to most. With the AbsoluteExpiration, the CacheItem is purged after a specified amount of time. With the SlidingExpiration, it is purged only if it has not been accessed after a specified amount of time. Using either mechanism is simply a matter of determining which one you want to use and picking an interval to use in conjunction with it.

The previous test used an AbsoluteExpiration; the following test shows a SlidingExpiration in action:

SlidingExpiration

```
[TestCase("Sliding1", 1)]
[TestCase("Sliding2", 2)]
[TestCase("Sliding3", 3)]
public void TestSlidingExpiration(string key, int value)
{
    // ARRANGE
    ObjectCache cache = MemoryCache.Default;
    CacheItemPolicy policy = new CacheItemPolicy
    {
        SlidingExpiration = new TimeSpan(0, 0, 2)
    };
    cache.Set(key, value, policy);

    // ACT
    for (var i = 0; i < 22; i++)
    {
        System.Threading.Thread.Sleep(100);
        Assume.That(cache.Get(key), Is.EqualTo(value));
    }
    System.Threading.Thread.Sleep(2001);

    // ASSERT
    Assert.That(cache.Get(key), Is.Null, "Uh oh!");
}
```

NOTE **THREAD.SLEEP**

In this last test, you can easily create a much faster test that tests the SlidingExpiration functionality and have it fail for all the wrong reasons. If you were to instead set the SlidingExpiration timer for one second and change nothing else in that test, you could very well see the Assume statements fail. The reason for this is unrelated to any caching concepts; it is that Thread.Sleep is a nondeterministic operation. What this means is that, in this for loop, it guarantees that it will pause execution for at least 100 ms, but it could even pause it for well over one second. It's even possible that you will see the published test fail for this reason on a slower or taxed system. Any time you use Thread.Sleep, always realize that it could actually pause execution for much longer than you anticipate, and if you depend upon it promptly resuming execution, you are setting yourself up for a race condition!

CacheItemPriority

Both the System.Web.Caching and System.Runtime.Caching namespaces have a CacheItem-Priority Enumeration that dictates the order of item removal from the cache. Both of these will be discussed because they might appear on the exam, but it's important to know the difference between them and that they are not the same enumeration even if they share the same name, same function, and many other characteristics. For now, the focus is on the System.Runtime.Caching namespace.

If no value is specified for the CacheItemPriority, the default value of Default is used. A value of Default means that there is No Priority. Although trying to memorize every member and every property of everything covered on the exam would be a challenging undertak-ing, memorizing each of the possible values of this version of the CacheItemPriority and the resulting behavior is so simple that I highly recommend doing so. See Table 1-2 to see each of the available values and what happens as a result of using the value.

TABLE 1-2 System.Runtime.Caching.CacheItemPriority enumeration value

Item name	Description
Default	There is no priority for removing this entry. As noted previously, this is the default value for the enumeration, so unless your intent is to make your code's intention clear, there's no value in setting this property and using this value.
Not Removable	Specifies that this entry should never be removed from the cache.

If you're thinking that the only two options seem to be to set no priority at all or set the priority in a way that ensures it's never automatically removed, you're correct.

One common source of confusion regarding CacheItemPriority (both the System.Runtime. Caching and the System.Web.Caching version) is that, by setting it, you could somehow create a situation in which the item can never be removed after it is inserted into the cache. That is not the case. Each version of the cache has methods specifically created for the removal of items from the cache. These values simply dictate the order of importance of the Cache Items when resources are low and the runtime needs to clean things up in order to continue processing.

Setting the value to NotRemovable can have very serious consequences, and you should clearly understand what they are before using this value. Setting the value to Default simply specifies that all CacheItems should be treated equally when a decision needs to be made to purge them because the system is getting low on resources. Striking the correct balance here is important because it is much better to take a painful database hit that causes an operation to be slow or even time out when the alternative is that the application crashes due to insufficient memory.

It is easy to overuse caching. There are many items that are clear candidates for caching and others that should almost never be placed into a cache. This determination is not always an easy one to make before an application goes to production with respect to each item you might choose to place in cache. Although this can be debated, it is often best to err on the side of not caching a particular set of data until you have the metrics necessary to prove that the data should be cached for performance reasons and can be done so without negatively affecting functionality.

If you choose the NotRemovable option, it is critical that you effectively monitor the object you're dealing with and take steps to ensure its removal from the cache explicitly. As previously mentioned, CacheItemPriority has no bearing on whether something can be removed manually; it simply prevents system optimization operations for purging it when trying to free up resources. If it stopped such removals from happening, anything put into cache with a CacheItemPriority would exist in cache for the life of the session, no matter what else happened. If you do not watch the items you marked as NotRemovable, you could easily find yourself in a situation in which the application's resources are completely consumed and it has no means by which to free up enough of them to keep functioning correctly.

The ChangeMonitor class

In addition to understanding the expiration policy, you need to be aware of the ChangeMonitor class, which the CacheItemPolicy has a collection of. If you refer to my example about the hospital intake application, it's quite obvious that, if the underlying data source changes, the cache needs to know about it. If it doesn't, there will be latency and a potential for very serious problems in many applications.

Although you don't use the ChangeMonitor class directly, it's a base class that has several derivations (and other ones can certainly be added):

- CacheEntryChangeMonitor
- FileChangeMonitor
- HostFileChangeMonitor
- SqlChangeMonitor

The CacheEntryChangeMonitor serves as a base class to build derived classes from that simply monitor the CacheItems for changes.

The FileChangeMonitor does exactly what its name implies: It monitors a specific file to see whether any changes are made to it, and, if so, the changes will be reflected in the cache, provided the CacheItemPolicy dictates it.

The HostFileChangeMonitor monitors directories and file paths for changes to them. If it detects changes, it responds accordingly. Because files and directories are fundamentally different from the other items, there are several specific items this class handles that you don't necessarily need to memorize for the exam, but should at least be familiar with. Each of the following triggers a change notification if a HostFileChangeMonitor is being used:

- The name of the monitored file or directory changed.
- The specified file or directory did not exist at the time the monitor was created, but was created later. In other words, a file or directory was created in the scope of the monitored items.
- The size of a monitored file changed.
- The contents of a monitored file changed, or the contents of a monitored directory changed.
- The *access control list (ACL)* of the file or directory changed.
- The monitored file or directory was deleted.

Using the HttpContext.Cache

ASP.NET applications (which are frequently used to host WCF Data Services) natively provide several different caching mechanisms. Because HTTP is a stateless protocol, state management is more of a challenge than it would be for a technology that used a different underlying protocol. In ASP.NET, you can use Application State, Session State, and View State to store values and thereby minimize roundtrips to the database. Each has strengths and weaknesses, and decisions about which to use should be undertaken carefully. Additionally, there are several options with regard to those features on what backing store can be used to host the data.

Because ASP.NET and applications hosted in Internet Information Server, for example, communicate using the HTTP protocol, it should come as no surprise that such applications are built in a way to include an HttpContext. The Cache class resides in the System.Web library and the System.Web.Caching namespace specifically. In a typical ASP.NET web application or service that's hosted in ASP.NET, the cache is accessed through the HttpContext.Current object. (The Page object also has a Page.Cache property and a Page.Context.Cache property; there are multiple ways to get to the Cache object.)

One of the most important things to understand about the Cache class is that it was built specifically for use with ASP.NET applications. If you need caching functionality in another type of application (for instance, a Windows Forms application), you should specifically use the ObjectCache class instead. (This might seem obvious, but more than a few developers have tried to add the System.Web.dll reference to Winforms apps or similar application types just so they could use the Cache feature.)

The following code snippet could certainly be added to a Console or Winforms application (provided that a reference to System.Web were added), but if you tried to execute it, it would result in a Null Reference Exception (and should be used only as an example of what *not* to do):

```
[Test]
public void BadCaching()
{
    // ARRANGE
    System.Web.Caching.Cache myCache = new System.Web.Caching.Cache();

    // ACT
    // ASSERT
    Assert.Throws<NullReferenceException>(() => myCache.Insert("asdf", 1));
}
```

If you want to pursue things that won't work and that you shouldn't do, you can try to add an HttpContext instance to a Winforms or Console app and then try to get the caching to work using it. The System.Web.Caching.Cache is meant to be used only with ASP.NET applications, not with other types of applications.

EXAM TIP

When taking the exam, you might see questions that specifically reference the HttpContext.Cache (or Page.Cache or some other similar form) directly. Such a question might be: "Datum Corporation has an application that's currently caching several collections of data in the HttpContext.Cache object. You need to _____. How should you do it?" In such cases, you can and should make the assumption that the application is an ASP.NET application or a WCF Data Service hosted inside of ASP.NET. Everything else that makes reference to caches (from determining what mechanism should be used the whole way to specific implementations) should be assumed to be referencing the ObjectCache unless specifically stated otherwise.

Another noteworthy item is that this class is created only once per AppDomain. Once created, it remains alive (even if it's empty) as long as the AppDomain is still active. There are several ways to add data to it, retrieve data from it, and control other aspects of its behavior, so let's go through them one at a time.

Cache items are implemented as name/value pairs where the name is implemented as System.String and the value is implemented as System.Object. You can put any serializable object in the cache (whether you should is a different decision altogether).

Abbreviated System.Web.Caching.Cache usage

The actual cache discussed is a property of HttpContext. You can refer to it directly as Cache["Key"] instead of HttpContext.Cache["Key"], which is done for the sake of readability.

NOTE **CACHE BEST PRACTICE**

Production code should minimize the use of quoted strings, particularly in cases where the string values are being used as keys to collections (such as caches). There are several reasons for this, including performance, maintainability, and ease of use. Defining the string literals using the const keyword at the class level is often appropriate, although in most caching scenarios, the string references will be made from several places. As such, it is often advisable to create the strings as constants or properties and keep them in classes of their own. Such approaches would add virtually no value from an exam preparation perspective, but would certainly add to the length of the code and decrease the readability. So the code samples throughout the chapter use quoted strings, but you shouldn't in any way infer that using them in this way is advisable or should be copied.

In its simplest form, you reference the cache, provide a key name, and give it a value:

```
Cache["FullNameKey"] = "John Q. Public";
```

This works in a manner reminiscent of View State or Session State. If there is already a key defined in the collection with the same name, the value you specify overwrites it. If no such key/value pair exists already, it simply adds it. Because the value part of the equation is typed as System.Object, you can set the value to just about anything.

In addition to the Add method, you can use the Insert method, which has several overloads; the simplest needs only a Key and a Value:

```
HttpContext.Current.Cache.Insert("AccountIdKey", new Account());
```

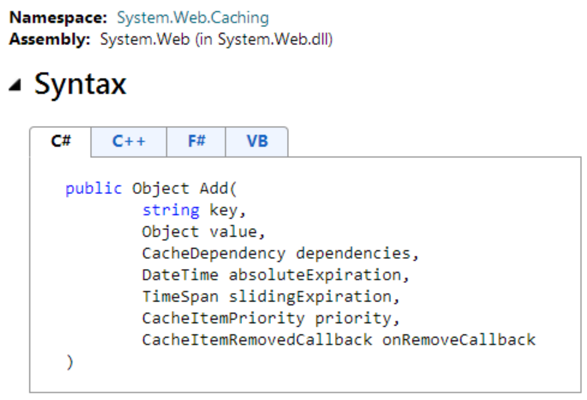
Table 1-3 shows each of the Cache.Insert overloads.

TABLE 1-3 Cache.insert overloads

Name	Description
Insert(String Key, Object Value)	Inserts an item into the cache with the corresponding key name and Object Value. The CacheItemPriority enumeration value is set to its default value, Normal.
Insert(String Key, Object Value, CacheDependency dependencyItem)	Inserts an item into the cache with the corresponding key name and Object Value that has file dependencies, key dependencies, or both.
Insert(String Key, Object Value, CacheDependency dependencyItem, DateTime absoluteExpiration, TimeSpan slidingExpiration)	Inserts an item into the cache with the corresponding key name and Object Value. This overload also includes file or key dependencies and expiration policies. When setting expiration policies, either the NoAbsoluteExpiration or NoSlidingExpiration predetermined value must be passed for the expiration policy not being used.

Name	Description
Insert(String Key, Object Value, CacheDependency dependencyItem, DateTime absoluteExpiration, TimeSpan slidingExpiration, CacheItemUpdateCallback updateCallback).	Inserts an item into the cache with the corresponding key name and Object Value. This overload includes key or file dependencies as well as expiration policies. It also includes a delegate object that can provide notification before an item is removed from the Cache collection.
Insert(String Key, Object Value, CacheDependency dependencyItem, DateTime absoluteExpiration, TimeSpan slidingExpiration, CacheItemPriority priority, CacheItemRemovedCallback removeCallback).	Inserts an item into the cache with the corresponding key name and Object Value. This overload includes key or file dependencies as well as expiration policies. It also includes a delegate object that can provide notification that the inserted item is removed from the Cache collection.

The initial example showed how to use the simplest method to add an item to the cache. The second showed an example of using the Insert method and provided a list of overloads and their corresponding signatures. Figure 1-7 shows the signature of the Add method of the cache.


Namespace: System.Web.Caching
Assembly: System.Web (in System.Web.dll)

Syntax

```
public Object Add(
        string key,
        Object value,
        CacheDependency dependencies,
        DateTime absoluteExpiration,
        TimeSpan slidingExpiration,
        CacheItemPriority priority,
        CacheItemRemovedCallback onRemoveCallback
)
```

FIGURE 1-7 System.Web.Caching.Cache Add signature

If you compare Add to the last version of the Insert method, you should notice quite a few similarities. In fact, they are virtually identical in just about every regard. For something to be added to the cache, you need a Key and a Value at a minimum.

If there are either file dependencies or dependencies to other cache keys, they are handled via the CacheDependency parameter. It’s worth a quick digression to mention this. Although there are several overloaded constructors for a CacheDependency, it is safe to describe them collectively as definitions for dependencies related to both cache keys and files (well, files or directories).

So the Insert method, just like the most comprehensive version of the Add method, accepts a string name for the Key, an object for the Value, CacheDependencies if they are to be included, a DateTime value that indicates an AbsoluteExpiration DateTime, a TimeSpan parameter to indicate how much time should elapse without an object being accessed before it is removed from the cache, a CacheItemPriority value to indicate where the given object falls on the removal hierarchy, and finally a delegate that can be used to notify the application if/when the corresponding CacheItem was removed.

In the final analysis, the only real difference between the two methods involves Update-Callback in the Insert method as opposed to CacheItemRemovedCallback employed in the Add method. They seem to be the same, so what's the noteworthy differentiator?

RemoveCallback (whose type is System.Web.Caching.CacheItemRemovedCallback) is called when an object is removed from the cache. UpdateCallback (whose type is System.Web.Caching.CacheItemUpdateCallback) executes a notification right before an object is removed from the cache. The behavioral difference can be exploited so that the cached item is updated and can be used to prevent the item from being removed from the cache. It's a subtle difference and one that you don't typically find necessary in many applications. Much of the time, when an item is removed from the cache, it is done so intentionally. However, expiration policy can cause objects to be removed from a cache without someone intentionally trying to remove it. Coupled with CacheItemPriority, scenarios can definitely be presented in which an item might be removed from cache contrary to the initial intention. When an item is different from others in terms of being automatically removed from the cache or when you need to be careful that it isn't removed (perhaps because retrieval of the object is quite expensive or resource-intensive), you might want to fine-tune behavior using the CacheItemUpdateCallback as opposed to the CacheItemRemovedCallback.

As mentioned earlier, both the System.Web.Caching and System.Runtime.Caching namespaces have a CacheItemPriority Enumeration that dictates the order of item removal from the cache. Although the one in the System.Runtime.Caching namespace is limited in what it provides, the one in the System.Web.Caching namespace affords a granular level of behavioral control.

The values of the enumeration are mostly self-explanatory because they do exactly what you expect, but Table 1-4 provides each available value and a description of the corresponding behavior. Just note that, although CacheItemPriority.Default was the behavior in System.Runtime.Caching version, a value of CacheItemPriority.Normal is the default value of the System.Web.Caching CacheItemPriority. Oddly enough, CacheItemPriority.Default here actually sets it to a Normal priority.

TABLE 1-4 System.Web.Caching.CachItemPriority enumeration values

Member	Description
Low	The lowest priority and therefore the first to be removed if the system starts removing items to free resources.
BelowNormal	The second least critical indicator of the enumeration. Only items marked as Low have lower priority.
Normal	The midpoint of what the system will remove. Lower in priority than High and AboveNormal, and more important than Low and BelowNormal.

Member	Description
AboveNormal	These items are considered less important than those specified as High, but more important than those specified as Normal.
High	These items are considered the most important and are the last to be removed when the system attempts to free memory by removing cached items.
NotRemoveable	Although this value stops something from being removed when resources are needed, it does not interfere with either the Absolute or Sliding expiration defined in the policy.
Default	Sets the value to Normal.

Using the CacheDependency class enables you to monitor and respond to a wide variety of changes to underlying objects. In most cases, you'll find that the CacheDependency can handle your needs quite well. If you are caching data that is sourced from a SQL Server database (versions 7.0, 2000, 2005, and later are supported), you can use the SqlCacheDependency class. SqlCacheDependency monitors the underlying table that the data originally came from. If any changes are made to the table, the items added to the Cache are removed, and a new version of the item is added to the cache.

To create a SqlCacheDependency, you can use either of two overloads:

1. You provide a SqlCommand instance. This initializes a new instance of the SqlCacheDependency class and, coupled with the command, creates a cache-key dependency.
2. You provide two strings. The first string is the name of the database defined in the databases element of the application's web.config file. The second string is the name of the table that the SqlCacheDependency will be associated with.

As you can imagine, creating such a dependency carries overhead and is laden with nuances that you need to be aware of. It is rare that you'll run into a situation in which some item outside of your application dictates what constructor must be used, but there are several "gotchas" related to the SqlCacheDependency in general. For instance, the version of SQL Server that's being used can dictate which constructor must be used.

It's worth noting that the baggage that comes with using the SqlCacheDependency is so significant that many consider it too substantial to justify the benefits. If you read through the MSDN documentation combined with what I call out in the following list, you'll find that there are many requirements that come with using the SqlCacheDependency, and a tremendous number of things need to be in place in order for it to be used correctly. If you know all of them, you know that there are several things that can result in not being able to use SqlCacheDependency for an application. Knowing these can certainly help you eliminate possible choices when it comes to answering test questions:

- The constructor that takes SqlCommand as a parameter is the one that should be used if you are using SQL Server 2005 or later. The fact that different constructors are supposed to be used based on the underlying database version is quite inconvenient in theory, but in practice most companies have at least migrated to SQL Server 2005 by now.
- With the SqlCommand-based constructor, two exceptions can be encountered: the ArgumentNullException and the HttpException. The first arises if you pass in a SqlCommand value that's null. This should come as no surprise because, without a valid command, it is impossible to determine what source the data came from. The second exception, however, is a little strange. It happens only when the SqlCommand passed to the constructor has the NotificationAutoEnlist value set to true *and* the @OutputCache directive is set on the page and it has a SqlDependency attribute value set to CommandNotification.
- Table names that are used in the CommandText property of SqlCommand must include the name of the table owner. So if your initial query's command text were "SELECT FirstName, LastName FROM Customer," the association would not work. If you use the same query but change the last portion to "... FROM dbo.Customer," the association should work if everything else is in place.
- Although using "SELECT *" is generally considered a poor practice from a performance perspective, using it actually breaks the functionality provided by the SqlCacheDependency. Using the previous query as an example, "SELECT * FROM dbo.Customer" would result in a failure.
- If page-level output caching is enabled on the page, this first constructor does not perform the association.

Although it might seem like this constructor carries with it a rather large number of requirements, those pale in comparison to the requirements and issues that surround the use of the second constructor:

- When using the second constructor, six different exceptions can be encountered (as opposed to just two when using the first one). These include HttpException, Argument Exception, ConfiguationErrorsException, DatabaseNotEnabledForNotificationsException, TableNotEnabledForNotificationsException, and ArgumentNullException. Table 1-5 provides a list of each of the exceptions and all the conditions that can trigger them.
- The connectionString that contains the table that the SqlCacheDependency is enabled for must be specifically included in the <connectionStrings> section of the web.config file of the ASP.NET application.
- The SQL Server database specified in the databaseEntryName parameter must have notifications enabled.
- The table specified in the tableName parameter must have notifications enabled as well.

- You cannot set monitoring up on multiple tables with one SqlCacheDependency. DataSets and most EF models make frequent use of related tables, and it seldom makes sense to cache a parent table and not any of the related tables. If you need this functionality, set up separate dependencies for each of the tables you are querying against and write application logic to deal the changes at each level in many cases. Suppose you have an Accounts table and a related AccountDetails table. If you wanted to implement a caching strategy, you'd probably want to cache both the Account information and the corresponding AcccountDetails information as well. At some point, however, any benefit realized by caching is offset by the overhead associated with the additional code required for cache monitoring. Finding that point is not always obvious and is often difficult.

TABLE 1-5 Exceptions related to second SqlCacheDependency constructor

Name	Description
HttpException	SqlClientPermission was not present or did not allow the operation access required. The DatabaseEntryName was not found in the list of databases. SqlCacheDependency could not make a connection to the configured database when the instance was initialized. The configured account lacked the permissions (either on the database or the stored procedures used internally to support SqlCacheDependency).
ArgumentException	The TableName parameter used to create the dependency association has a value of String.Empty.
ArgumentNullException	DatabaseEntryName passed in to the constructor was null. The TableName parameter used to create the dependency association was null. (Yes, this is different than if the parameter were an empty string.)
ConfigurationErrorsException	SqlCacheDependency does not have Polling enabled. Polling is enabled, but the interval is not correctly configured. There was no connectionString matching the parameter name in the <connectionStrings> section of the configuration file. The connectionString specified in the configuration file could not be found. The connectionString specified in the configuration file contained an empty string. Although the MSDN documentation doesn't mention it specifically, if the connectionString is configured in an invalid format or can't be parsed, a ConfigurationErrorsException is thrown as well.
DatabaseNotEnabledForNotificationException	The configured database entry indicated by the databaseEntryName parameter does not have change notifications enabled.
TableNotEnabledForNotifications	The name of the table specified in the tableName parameter does not have change notifications enabled.

Using a SqlCacheDependency can be a daunting task. Spend the time to configure everything necessary to actually see SqlCacheDependency in action; doing so will certainly be educational.

Thought experiment

Develop a caching strategy

In the following thought experiment, apply what you've learned about the "Implement caching" objective to predict what steps you need to develop an effective caching strategy. You can find answers to these questions in the "Answers" section at the end of this chapter.

Contoso has determined that its primary database is experiencing tremendous traffic volumes. Upon analysis, it determines that much of the traffic is around identical requests.

With this in mind, answer the following questions:

1. **What type of data would you consider as a candidate for caching?**
2. **What factors would you consider when determining a caching strategy?**
3. **What would you contemplate regarding data changes?**

Objective summary

- Although caching data isn't a magic wand to fix performance problems, most applications have very obvious aspects that lend themselves well to data caching.
- ObjectCache is the primary mechanism you can use to cache data.
- The Cache property of HttpContext can be used to provide caching functionality in ASP.NET applications.
- When using ObjectCache, the two most high profile elements are ExpirationPolicy and ChangeMonitoring.
- A specific date and time can trigger cache expiration, which is known as AbsoluteExpiration. For instance, by using AbsoluteExpiration, you can wipe out the cache or remove an item from it at midnight every day, once an hour, or at whatever time makes sense.
- Expiration can be handled so that an object is removed from the cache only if it has not been accessed for a certain specified interval using SlidingExpiration. For instance, you can remove an item from the cache if it has not been accessed for two hours.

Objective review

Answer the following questions to test your knowledge of the information discussed in this objective. You can find the answers to these questions and their corresponding explanations in the "Answers" section at the end of this chapter.

1. Which of the following are provided for use by the .NET Framework as ChangeMonitors when defining a CacheItemPolicy? (Choose all that apply.)
   - **A.** CacheEntryChangeMonitor
   - **B.** FileChangeMonitor
   - **C.** MsmqChangeMonitor
   - **D.** SqlChangeMonitor
2. Which values are valid choices when defining the Priority property of the CacheItem-Policy using the System.Runtime.Caching version of the Cache? (Choose all that apply.)
   - **A.** Normal
   - **B.** High
   - **C.** NotRemovable
   - **D.** Low
3. You have set up an ObjectCache instance using the following code:

```
List<String> fileList = new List<String>();
fileList.Add(@"C:\SomeDirectory\SampleFile.txt");
ObjectCache cacheInstance = MemoryCache.Default;
CacheItemPolicy accountPolicy = new CacheItemPolicy();
accountPolicy.Priority = CacheItemPriority.Default;
accountPolicy.AbsoluteExpiration = DateTime.Now.AddMinutes(60);
accountPolicy.ChangeMonitors.Add(new HostFileChangeMonitor(fileList));
CacheItem exampleItem1 = new CacheItem("ExampleItemId", "Example Item Value",
"AccountObjects");
```

   Which of the following items add an item to the cache with a key named "ExampleIt-emId," a value of "Example Item Value," a region named "AccountObjects," and a CacheItemPolicy with a Default CacheItemPriority? (Choose all that apply.)
   - **A.** cacheInstance.Add(exampleItem1, accountPolicy);
   - **B.** cacheInstance.Add(exampleItem1, accountPolicy.Priority.Default);
   - **C.** cacheInstance.Add("ExampleItemId", "Example Item Value", accountPolicy);
   - **D.** cacheInstance.Add("ExampleItemId", "Example Item Value", accountPolicy, "AccountObjects");

Objective 1.3: Implement transactions

Once upon a time, flat *indexed sequential access method (ISAM)* databases ruled the world, and transactions weren't a big deal and were seldom even a concern on most developers' minds. Today, with e-commerce booming and even traditional commerce being largely computerized, the need for transactions shows itself frequently in applications (and this should in no way imply that transactions are only relevant to commerce).

Transactions are powerful and, when used correctly, provide tremendous functionality commensurate to the effort it takes to use them. At the same time, transactions are not free in terms of resource utilization, and incorrect implementations have been the source of countless headaches for database administrators, developers, salespeople, and end users alike.

For this exam, you should be familiar with how to effectively implement a transaction with ADO.NET, the EF, and general characteristics of transactions. System.Transactions is the core namespace that facilitates generic transaction functionality. You also need at least a basic understanding of EntityTransaction (located in the System.Data.EntityClient namespace) and the SqlTransaction class.

This objective covers how to:

- Understand the characteristics of transactions
- Implement distributed transactions
- Specify a transaction isolation level
- Use the TransactionScope
- Use the EntityTransaction
- Use the SqlTransaction

Understanding characteristics of transactions

To meet the technical criteria for a database transaction, it must be what's known as *ACID*, which is an acronym for atomic, consistent, isolated, and durable. Entire books and research papers have been written on the subject, so being an expert in transaction theory is not necessary (or even necessarily helpful) for this exam, but it will help you understand why the specific implementations that are covered on the exam operate as they do.

Although referencing Wikipedia is something that must be done with extreme caution, the content in one Wikipedia article is good. It states that transactions have two primary purposes:

> *To provide reliable units of work that allow correct recovery from failures and keep a database consistent even in cases of system failure, when execution*

stops (completely or partially) and many operations upon a database remain uncompleted, with unclear status.

To provide isolation between programs accessing a database concurrently. If this isolation is not provided, the program's outcome, are possibly erroneous.

Transactions serve to make database interactions all-or-nothing propositions. Once executed, they need to complete successfully, or they need to be completely undone and leave things exactly as they were had the change attempt not been made if they failed. If you understand that, you understand the fundamental concept of transactions.

One aspect of transactions that you probably need to know for the exam is the concept of *isolation levels*. To be of much value, one transaction needs to be kept from, or isolated from, all other transactions. SQL Server has extensive support for both transactions and isolation levels, so they are likely to appear on the exam.

The other concept that you will likely encounter is that of *simple transactions* versus *distributed transactions*. A simple transaction is the most common-use case you've probably heard of. An application attempts to insert a record into an Account table and then an Account Details table. If either fails, both should be undone, so there aren't any orphaned records or incorrectly recorded items. Simple transactions affect one database, although multiple tables might be involved. Other cases, and this is particularly relevant in today's world of large distributed networks, are distributed transactions that span multiple systems.

Implementing distributed transactions

It's not uncommon for one company to purchase software products for specific functionality that's written by different vendors. They often write an application that consolidates critical information from each system into one place or that tries to get the applications to work together. Proprietary software that wasn't designed to work with external systems was once the norm, but that time has passed, and customers increasingly find such self-serving functionality unacceptable. In these cases, transactions become important.

Imagine a system that handles front-end cash register purchases, and imagine another one that records the purchases and returns in the accounting system. If a purchase is made and successfully recorded in the register system but it fails to record the transaction in the ledger system, a problem results. If the reverse is true, it is also a serious problem.

Now imagine that you also incorporate an inventory system in the equation that automatically handles ordering things when certain thresholds are set. If this system goes down for 30 minutes, if the purchases are made and recorded correctly, and the items are accounted for in the accounting system correctly, but the inventory system has no idea any of these sales just happened, it would think that many things still existed in inventory that didn't. So it wouldn't order more of them to replenish inventory just yet. You can see how this can have serious consequences for the company.

The problem is that, in a case like this, you'd likely be dealing with three separate applications written by three separate companies, so the simple transaction model wouldn't work.

That's because in a simple transaction scenario, one database connection is used. In this case, you have three separate ones, and it's possible that those three databases aren't even of the same type. Although this in no way indicates a deficiency on the part of the developers of the System.Data.SqlClient.SqlTransaction class, it should come as little surprise that it does not provide support for an Oracle database, a Sybase database, or a CouchDB database. It doesn't mean that they aren't sufficient or couldn't be used individually on each machine, but that wouldn't do much to solve the problem at hand.

To address these scenarios, Microsoft made a significant investment in building solid tools to deal with them using distributed transactions. These tools will be discussed in depth and, outside of a limited number of questions on basic transactions, it's likely you'll see at least some questions on the exam related to distributed transactions. The core Transactions objects exist in in the System.Transactions namespace. The two other relevant ones are the System.Data.SqlClient and System.Data.EntityClient namespaces (the latter being from the System.Data.Entity assembly).

Specifying a transaction isolation level

The IsolationLevel enum is used to manage how multiple transactions interact with one another. Another way to describe it is that IsolationLevels control the locking behavior employed for the execution of a command. However, there's a problem that many developers stumble over, so let's first get this out of the way. There are actually two Isolation enums: one in System.Data.IsolationLevel and a second in System.Transaction.IsolationLevel. Just as there were two CacheItemPriority enums that generally served the same purpose, these two IsolationLevel enums generally serve the same purpose, but for two different sets of classes. Fortunately, both have the same values, so there isn't much to remember between the two other than the fact that the two exist; sometimes you need one, and other times you need the other.

EXAM TIP

The IsolationLevel enumeration values have not changed since the enumeration was introduced initially. Table 1-6 covers each of the values and what they do, but you would be well advised to learn each of these and understand them. The official documentation for each behavior is available on MSDN at the following URL: *http://msdn.microsoft.com/en-us/library/system.data.isolationlevel.aspx*. Because questions regarding IsolationLevel are very likely to appear on the exam, by understanding what each level does, you'll be able to distinguish the correct answer based on requirements mentioned in the question. You'll likely see something in a question stub indicating that you need to allow or prevent exclusive range locks or you need to ensure that users are prevented from reading data locked by other transactions. Such verbiage is a dead giveaway to which isolation level is correct.

Table 1-6 lists each value of the enumeration and describes the implications of using it according to MSDN:

TABLE 1-6 System.Data.IsolationLevel

Member	Description
Unspecified	The actual transaction level being used cannot be determined. According to MSDN, if you are using an OdbcConnection and do not set this value at all, or you do set it to Unspecified, the transaction executes according to the isolation level that is determined by the driver that is being used.
Chaos	The pending changes from more highly isolated transactions cannot be overwritten. This is not supported in SQL Server or Oracle, so it has very limited use.
ReadUncommitted	No shared locks are issued; exclusive locks are *not* honored. The important implication is that this isolation level can result in a dirty read, which is almost always undesirable.
ReadCommitted	Shared locks are held during reads. This has the result of avoiding dirty reads, unlike ReadUncommitted. However, the data can be changed before the end of the transaction, resulting in nonrepeatable reads or phantom data.
RepeatableRead	Locks are placed on all data used in the query, which completely prevents others from updating any data covered by the lock. It stops nonrepeatable reads, but the phantom data problem is still possible.
Serializable	A range lock is placed specifically on a DataSet. No one else can update the data or insert rows into the set until the transaction is completed. Although very powerful and robust, this state can cause major problems if it is not used quickly.
Snapshot	An effective copy of the data is made, so one version of the application can read the data while another is modifying the same data. You can't see data from one transaction in another one, even if you run the query again. The size of them can also cause problems if overused.

NOTE **CHANGING THE ISOLATIONLEVEL DURING EXECUTION**

As you look at the constructors of classes such as EntityCommand, SqlCommand, TransactionScope, and many other data classes, you'll notice they each have the ability to specify a transaction. Although there are not many use cases you'll typically run across, you might encounter a situation in which a different IsolationLevel is desired for different phases of the transaction's execution. The default IsolationLevel of one set initially remains in effect for the life of the transaction, unless it is explicitly changed. It can be changed at any time the transaction is alive. The new value takes effect at execution time, not parse time. So if the IsolationLevel is changed somewhere midstream in execution, it applies to all remaining statements.

Managing transactions by using the API from the System.Transactions namespace

The TransactionScope class was introduced in version 2.0 of the .NET Framework. It's easy to use and powerful. Other than the declaration and instantiation of it, the only thing you need to know about it is that it has a method named Complete() that you should call if you are satisfied it completed successfully. This is a key point. Calling Complete() tells the transaction manager that everything should be committed. If it isn't called, the transaction is automatically rolled back. Also, when called correctly in a using block, if an Exception is thrown during execution inside the TransactionScope, the transaction will be rolled back as well.

Here's a nonfunctional sample (connections and commands aren't what they should be) of how to use the TransactionScope in conjunction with a SqlConnection:

```
using (TransactionScope mainScope = new TransactionScope())
{
    using (SqlConnection firstConnection = new SqlConnection("First"))
    {
     firstConnection.Open();
     using (SqlCommand firstCommand = new SqlCommand("FirstQueryText", firstConnection))
        {
            Int32 recordsAffected = firstCommand.ExecuteNonQuery();
        }
        using (SqlConnection secondConnection = new SqlConnection("Second"))
        {
            secondConnection.Open();
            using (SqlCommand secondCommand = new SqlCommand("SecondQueryText",
secondConnection))
            {
                Int32 secondAffected = secondCommand.ExecuteNonQuery();
            }
        }
    }
    mainScope.Complete();
}
```

Besides its simplicity, it also has the benefit of being able to handle both simple and distributed connections and promote simple transactions to distributed ones automatically. In the previous example, a new TransactionScope was declared and instantiated. Then two SqlConnections were created inside of it and two corresponding SqlCommands. There are no exception handlers, so any exceptions result in the transaction being rolled back.

There are two important takeaways here. First, when the call to Open() is made on FirstConnection, it's created inside a simple transaction. When Open is called on SecondConnection, the transaction is escalated to a full distributed transaction. This happens automatically with no intervention on the developer's part. The second takeaway is that, in order for everything to happen correctly, the last statement, the call to Complete(), must happen before the transaction commits.

Distributed transactions are not simple in just about any regard, even if the TransactionScope makes it look easy. A lot has to happen in order for them to work correctly. People frequently assume that they can do anything inside a TransactionScope (such as copy files or call Web Services), and if Complete isn't called, it's all rolled back. Although Web Services can be created to support transactions, it doesn't just happen automatically. In the same respect, not all databases support transactions, so if one is being used that doesn't, there's not much that can be done about it. Another obvious example is that, if you send an e-mail in the middle of a TransactionScope, you cannot undo sending that e-mail just because you don't call Complete on your TransactionScope later.

It's doubtful that much would be covered in the exam with respect to the TransactionScope outside of what has been covered already: know to call Complete(), know that promotion happens automatically, know that some additional requirements are there if you want to have distributed transactions, and know that exceptions result in a rollback if you use the TransactionScope inside a using block. Focusing on those aspects is much more fruitful than trying to memorize the complete list of items supported by the distributed transaction coordinator.

Using the EntityTransaction

The main purpose of this class is to specify a transaction for an EntityCommand or to use in conjunction with an EntityConnection. It inherits from the DBTransaction base class. The EntityTransaction class has two main properties to be concerned with: the Connection property and the IsolationLevel property. It has two methods of primary concern as well: Commit() and Rollback(). There are a few other methods, such as Dispose(), CreateObjReference(), ToString(), and some others, but they are unlikely to appear on the exam.

One important note is that, when trying to implement a transaction within the EF, it isn't necessary to explicitly use the EntityTransaction class (or TransactionScope, for that matter). Look at the following code:

```
 using (TestEntities database = new TestEntities())
{
    Customer cust = new Customer();
    cust.FirstName = "Ronald";
    cust.LastName = "McDonald";
    cust.AccountId = 3;
    database.Customers.Add(cust);
    database.SaveChanges();
}
```

Although it might not be obvious (especially because this is a section on the EntityTransaction class), the SaveChanges method of the DbContext automatically operates within the context of a transaction. When the SaveChanges() method is called, each item in the current context instance is examined for its EntityState. For each record that has a status of Added, Deleted, or Modified, an attempt is made to update the underlying store corresponding to the EntityState. If any of the items can't be saved back to the source, an exception is thrown, and the transaction that the changes were executing in is rolled back.

Of course, there are times when you are using a different approach or need to explicitly manage the transactions yourself. If you are in a situation in which you are using an EntityCommand instead of the ObjectContext, you can use the EntityTransaction almost identically to the way you use the SqlTransaction: You simply create a new EntityConnection, declare a new EntityTransaction by calling BeginTransaction, and then perform your logic. Based on the results, you can choose to call the Commit method or the Rollback method.

```
using (EntityConnection connection = new EntityConnection("TestEntities"))
{
    using (EntityTransaction trans = connection.BeginTransaction(System.Data.
IsolationLevel.Serializable))
    {
        EntityCommand CurrentCommand = new EntityCommand("SOME UPDATE STATEMENT",
connection, trans);
        connection.Open();
        Int32 RecordsAffected = CurrentCommand.ExecuteNonQuery();
        trans.Commit();
    }
}
```

Using the ObjectContext and SaveChanges is usually more than sufficient for most application needs, and the TransactionScope can equally handle most cases when transactional functionality is needed. If you need to use the EntityCommand specifically, however, you can use the EntityTransaction, as shown previously.

Using the SqlTransaction

If you look at the last few paragraphs, the discussion there is virtually identical to what's covered here. The behavior of the SqlTransaction is identical; to perform the same scenario, the only things that changed were the names of the objects and their types.

You create a SqlConnection, call the BeginTransaction() method specifying an IsolationLevel, create a SqlCommand setting the CommandText Property (or Stored Procedure name and changing the CommandType property), add a SqlConnection to it, and pass in a SqlTransaction as the last parameter. Then you perform whatever action you want on the SqlCommand instance, and call Rollback or Commit when you're done, depending on the outcome of the execution.

Thought experiment

Working with transactions

In the following thought experiment, apply what you've learned about the "Implement transactions" objective to predict what steps you should take to effectively manage transactions. You can find answers to these questions in the "Answers" section at the end of this chapter.

Contoso has several mission-critical processes that are starting to cause problems. Users are reporting increasingly frequent processing errors. Additionally, there are several problems with partial or incomplete records being created. For instance, users are receiving Order Processing Failure messages and then getting tracking numbers for their order.

With this in mind, answer the following questions:

1. **What would make an operation a good candidate for a transaction?**
2. **Why not make every operation transactional?**
3. **What considerations would guide the choice of transaction implementation?**

Objective summary

- There are several ways to implement database transactions in the current .NET Framework, including using the EF SaveChanges method, the EntityTransaction, the SqlTransaction, and the TransactionScope.
- TransactionScope is often the quickest and easiest way to work with transactions in .NET.
- IsolationLevel is common to every form of transaction and is the one thing that has the biggest effect on transaction behavior. Its importance is often taken for granted, but it is probably the most important aspect of any transaction besides committing and rolling back.
- Transactions can take two forms: simple and distributed. You can often identify distributed transactions by seeing multiple or different types of connection objects.

Objective review

Answer the following question to test your knowledge of the information discussed in this objective. You can find the answers to this question and its corresponding explanation in the "Answers" section at the end of this chapter.

1. You are developing an ASP.NET application that reads data from and writes data to a SQL Server database. You should avoid nonrepeatable reads and not concern yourself with phantom data. Which isolation level should you use?
   - **A.** ReadUncommitted
   - **B.** RepeatableRead
   - **C.** Chaos
   - **D.** Snapshot
2. Which items would benefit from being made transactional?
   - **A.** Long-running queries that span multiple tables
   - **B.** Quick-running queries that involve only one table
   - **C.** Queries that make use of the file system on a Windows operating system.
   - **D.** Queries that make use of the file system of a non-Windows operating system.

Objective 1.4: Implement data storage in Windows Azure

You wouldn't be taking this test if you weren't at least nominally familiar with Web Services and the cloud. Windows Azure is a big area, and there are several books on it. This objective specifically relates to data storage in Windows Azure. Although learning all about other aspects of Windows Azure is great, keep focused on the fact that you are just dealing with data storage for this particular objective.

This objective covers how to:

- Access data storage in Windows Azure
- Choose a data storage mechanism in Windows Azure (blobs, tables, queues and SQL Database)
- Distribute data by using the Windows Azure Content Delivery Network (CDN)
- Manage Windows Azure Caching
- Handle exceptions by using retries (SQL Database)

Accessing data storage in Windows Azure

Data storage can mean a lot of things and cover a lot of areas, so let's be specific. You should make sure that you fully understand the storage options that are available and their respective strengths and weaknesses. You should understand how to read and write data using these methods. There are several other objectives in other chapters that relate to multiple other aspects of Windows Azure, but for this portion, keep focused on data storage.

Whether you're pondering it in the real world or just for the exam, when deciding about data storage in Windows Azure, it needs to be compared against something. What is that something? You can come up with all sorts of options: a USB flash drive, an SD card, a USB or SATA drive, a traditional hard drive, or all of these in a storage area network. But that's the wrong context. No one is running a production SQL Server and storing the data on a USB card. Because this is a Microsoft-specific exam, you can also rule out non-Microsoft solutions. So data storage with Windows Azure versus data storage in a Windows Server environment is what you need.

You can deduce that, if Windows Azure data storage options didn't work closely to existing mechanisms, it would greatly impede adoption. Few people have gotten fired for playing it safe and doing what always worked (although they often sit by and watch companies slowly die of attrition). Countless numbers of people have been fired for pushing some new technology that results in failure. You can be sure that a company such as Microsoft is quite aware of such dynamics. It is also aware of the fact that people focus on total cost of ownership. So any benefits from Windows Azure will be weighed against costs and compared against the existing storage solution. There's nothing Windows Azure could offer that would convince many companies to adopt it if doing so meant porting all their existing apps to a brand new storage mechanism that was totally different from what they have now. So things, for the most part, map to existing technologies quite well, and in many cases can be migrated transparently.

But everyone who has ever written software and lived through a few versions is aware of how challenging issues such as breaking changes and legacy anchors can be. It's impossible to make any substantive improvements without some risk. And sometimes you have no choice but to introduce breaking changes and end support for items. Similarly, cloud-based storage, for instance, has to have some other benefits and features other than "someone else can worry about our data." That's a verbose way of saying that, although much of what you'll encounter in Windows Azure data storage is identical to that of Windows Server storage, there are a few things that are notably different or completely new.

Finally, following are some significant concerns to be aware of when implementing data storage in Windows Azure:

- Applications ported to the Windows Azure platform are dependent on network access. Lose your Internet connection and you have problems. This is a lot different from the typical in-house application scenario, in which you can take consistent access to resources such as the database or a message queue for granted.
- If you use local storage with Windows Azure, existing code and methodologies are almost identical. It's only when you are dealing with Table and Blob storage (introduced shortly) that you see access methods different from what you're familiar with.

You are dealing with a large layer between your application and the data store. The whole reason to use cloud-based options is so you no longer have to worry about data storage or uptime of the apps; someone else does. But that doesn't magically make things always work. A *Service Level Agreement (SLA)* or Uptime Agreement might promise the mythical 5-9s, but that doesn't mean it is so. It just means you have recourse if the expectations aren't met. Just as you don't have unlimited processor utilization, you also can't take always-available Internet for granted.

Depending on how you want to count things, there are either five or three storage offerings in Windows Azure. Table 1-7 shows these offerings. You can consider Windows Azure storage as an offering, or you can consider each individual component as an offering. At the highest level, you have Local Storage, Windows Azure Storage, and SQL Database as options (three). If you count the components of each, you have Local Storage, Blob, Table, and Queue (all of which are part of Windows Azure storage), and SQL Database.

TABLE 1-7 Windows Azure platform storage options

Offering	Purpose	Capacity
Local Storage	Per-instance temporary storage	<20 GB–2 TB
Blob	Durable storage for large binary objects (audio, video, image files)	200 GB–1 TB (1 TB per page Blob, 100 TB per account)
Table	Tabular or structured data	100 TB
Queue	Items that you normally store in a Message Queue (MSMQ)	100 TB
SQL Database	Exactly what you expect: an online version of SQL Server	150 GB

Please make sure that you understand the differences between these options. There's also additional detail available at *http://social.technet.microsoft.com/wiki/contents/articles/1674.data-storage-offerings-on-the-windows-azure-platform.aspx*, which you will likely find helpful.

For the exam, it is likely that you will run into a situation in which you are given a scenario and then asked what items fit each piece of it. When you read Video, Audio, Image, it's almost certain to be a candidate for Blob storage. When it's structured data of any format, it will be Table storage or SQL Database storage if the structured data is also relational. Queue storage is typically hinted at by something along the lines of needing a guarantee of message delivery. SQL Database items should be clear enough to denote in and of themselves. That leaves Local storage. The clue to it is that it is temporary storage, it mimics your typical file system, and it is per-instance.

Choosing a data storage mechanism in Windows Azure (blobs, tables, queues and SQL Database)

At the most basic level, blobs, queues, and tables differ in the following ways:

- **Blobs** are ideally suited for unstructured binary and text data. There's nothing stopping you from breaking apart an Excel file and storing it as binary in Blob storage. There's nothing even stopping you from taking a SQL Server or Oracle data file, reading the bytes and storing it as a blob. But it makes little sense to do so because there are much better ways to store structured data (just in case you missed it, each of the aforementioned items is considered a traditional "structured" data store). In the same respect, though, storing backups or copies of these files is something you might typically consider doing. You can store structured data in Blob storage, but the cases in which you'll need or want to do it are few if any.
- **Tables** are ideally suited for storing structured but nonrelational data. If you're familiar with them, think NoSQL databases such as CouchDB or MongoDB. Suppose that you were developing a YouTube-like service that enabled users to upload videos, and you wanted to store the metadata of the video, the user, and so on. You'd typically want to store the videos in Blob storage and store the other metadata in Table storage (which might entail multiple tables or just one).
- **Queues** are essentially the Windows Azure equivalents of an MSMQ. If you're unfamiliar with Queues or MSMQ, do a quick search and become familiar with them. A queue is a First In, First Out (FIFO) data structure that can store almost anything that can be serialized. Queues are ideally suited for situations in which delivery of a message or processing of information absolutely must happen. It's similarly well suited to operations characterized by long-running processes and asynchronous jobs. There is a specific WCF Binding to use for queues that is the primary way reliable messaging is facilitated with Windows Azure.

Bindings and reliable messaging haven't been introduced yet, but if you're unfamiliar with them, an overview is provided in the following coverage.

Blob storage

Blob storage enables you to retrieve and store large amounts of unstructured data. One of the main benefits is that, once stored, the corresponding information can be accessed from anywhere using the HTTP or HTTPS protocols. If you have Internet access, you can retrieve and manipulate these items. The items you store in Blob storage can range in size up to 100 TB, although the more likely scenario is that you will deal with several smaller items.

Recall that you can store SQL Server data files or Oracle data files in Blob storage, but it makes little sense to do so because you have the option to use SQL, Windows Azure, and Table storage. At the same time, it makes perfect sense to store your backups for them in Blob storage. Why one and not the other? Because Blob storage is ideally suited to large sizes and unstructured data. The data file for a production database needs to be written to and read from constantly, whereas the backup just needs to be uploaded and accessed if and when needed. Blob storage is ideally suited for the following:

- Images that can be directly viewed in a browser
- Document storage
- Secure backups as part of a disaster recovery plan
- Streaming video and audio

The structural makeup of Blob storage needs to be discussed at this point. At the highest level, you have the storage account. Storage accounts hold containers. Containers in turn contain blobs.

At the top of the entire storage structure is the storage account. Without it, you have nothing. You'll notice that in Table 1-8, there is a 1 terabyte (TB) limit on Page Blob size, but there's a 100 TB limit on a storage account. If your data storage exceeds 100 TB, you have to handle everything notably differently than you would otherwise. If you are dealing with that much data, however, there will be some complexities and difficulties irrespective of storage mechanism. A storage account's main purpose with respect to blobs is to be a holding mechanism for containers. In practical terms, there's no limit on the number of containers that a storage account can hold (as long as they don't exceed the size limit).

Although hardly a perfect metaphor, think of a container as a subdirectory used to organize your blobs within a storage account. Although containers cannot be nested within one another, they can be used in many different ways for organizational purposes. A storage account can contain any number of containers, provided that the sum of each container's size doesn't exceed 100 TB. Containers have one essential task: to provide a logical and physical grouping for blobs.

A blob is merely a file. There are no type restrictions, so they can hold almost anything. Windows Azure has two distinct categories of blobs:

- Blocks
- Pages

Block blobs have a 200 gigabyte (GB) size limit. Page blobs, on the other hand, can store anything up to 1 TB.

You have two choices for accessing blob data. The first is that you can just reference it by URL. The structure of the URL uses the following format:

```
http://<storage account name>.blob.core.windows.net/<container>/<blob>
```

The next method involves using the API. To reference these namespaces, you have to add a project reference to the Microsoft.WindowsAzure.Storage.dll assembly. The API for Blob storage includes the following namespaces:

- Microsoft.WindowsAzure.Storage
- Microsoft.WindowsAzure.Storage.Auth
- Microsoft.WindowsAzure.Storage.Blob

Ensure that you have a storage account created within your Windows Azure subscription. The storage account is represented by the CloudStorageAccount class. It's instantiated in a few ways. First, you can use the two constructors, which work in one of two ways. The first overload takes an instance of StorageCredentials as the first argument and then a *uniform resource identifier (URI)* representing the blob endpoint, Queue endpoint, or Table endpoint, in that order. If you're using the Blob, as in this case, just leave the other values null. You do the same sort of thing if you are using just a queue or just a table. The second one works similarly: Specify an instance of the StorageCredentials and whether to use HTTPS. You need to set the values for the URIs in your code afterward or manually configure them through the Windows Azure UI:

First Blob storage upload

```
[TestCase("1234", "count.txt", "file", "INSERT YOUR ACCOUNT NAME", "INSERT YOUR KEY")]
public void UploadBlob(string fileContents, string filename, string containerName,
string accountName, string accountKey)
{
    // ARRANGE
    StorageCredentials creds = new StorageCredentials(accountName, accountKey);
    CloudStorageAccount acct = new CloudStorageAccount(creds, true);
    CloudBlobClient client = acct.CreateCloudBlobClient();
    CloudBlobContainer container = client.GetContainerReference(containerName);

    // ACT
    container.CreateIfNotExists();
    ICloudBlob blob = container.GetBlockBlobReference(filename);
    using (MemoryStream stream = new MemoryStream(Encoding.UTF8.GetBytes(fileContents)))
    {
        blob.UploadFromStream(stream);
    }

    // ASSERT
    Assert.That(blob.Properties.Length, Is.EqualTo(fileContents.Length));
}
```

A much better way to do this is using the static Parse or TryParse methods of the CloudStorageAccount. You have two choices with either approach. You can use the specific CloudConfigurationManager and call the GetSetting method, which pulls the setting from the .CSCFG file associated with your compiled Windows Azure Cloud project. Or you can use the ConfigurationManager from the System.Configuration assembly; reference the name key as a parameter; and use the ConnectionStrings property, which will pull the setting from the application's web.config or app.config file.

Following is an example that pulls the connection info from the test project's app.config file:

```
[TestCase("1234", "count.txt", "file")]
public void UploadBlobFromConfig(string fileContents, string filename, string
containerName)
{
    // ARRANGE
    CloudStorageAccount acct = CloudStorageAccount.Parse(ConfigurationManager.Connection
Strings["StorageConnection"].ConnectionString);
    CloudBlobClient client = acct.CreateCloudBlobClient();
    CloudBlobContainer container = client.GetContainerReference(containerName);

    // ACT
    container.CreateIfNotExists();
    ICloudBlob blob = container.GetBlockBlobReference(filename);
    using (MemoryStream stream = new MemoryStream(Encoding.UTF8.GetBytes(fileContents)))
    {
        blob.UploadFromStream(stream);
    }

    // ASSERT
    Assert.That(blob.Properties.Length, Is.EqualTo(fileContents.Length));
}
```

To get this to work, you need the following line in your app.config file, slightly tweaked with your settings:

```
   <add name="StorageConnection" connectionString="DefaultEndpointsProtocol=https;Account
Name=ACCOUNT_NAME_GOES_HERE;AccountKey=ACCOUNT_KEY_GOES_HERE" />
```

You can, and probably should, use the TryParse method in a similar fashion. The debate between using Parse versus TryParse is out of the scope of this book.

After you build up a storage account reference, you need to create a client using the CloudBlobClient class. Next, get a container reference, ensure that it's been created, get your blob reference, and save it. Downloading blobs is just as easy when using the DownloadToStream method from the ICloudBlob object.

Remember that blobs come in two forms: Block blobs and Page blobs. As such, there's a CloudBlockBlob class and a CloudPageBlob class to represent either of them. You retrieve either of them by using the CloudBlobContainer class and calling the GetBlockBlobReference method or the GetPageBlobReference method passing in the name of the item, respectively. They both work the exact same way as the prior tests.

The next thing is how to write to a blob. Once you have a reference to a CloudBlockBlob or CloudPageBlob, you need to call the UploadFromStream method, passing in a FileStream or some other Stream reference. There are several overloads and asynchronous methods to accompany it, and it's a good idea to do a quick review of them when preparing for the exam. Uploading huge amounts of data isn't usually the fastest process in the world, so you typically should make sure that you use asynchronous methodology so you don't block the primary thread of your application.

You should wrap the call to the UploadFromStream in an exception handler and respond to any problems that you might encounter. Keep in mind that UploadFromStream overwrites data if it is already there, so that's the mechanism for updating existing blobs, too. Deleting blobs is just a matter of calling the Delete or DeleteIfExists methods of the respective items:

```
[TestCase("count.txt", "file")]
public void DeleteBlobFromConfig(string filename, string containerName)
{
    // ARRANGE
    CloudStorageAccount acct = CloudStorageAccount.Parse(ConfigurationManager.Connection
Strings["StorageConnection"].ConnectionString);
    CloudBlobClient client = acct.CreateCloudBlobClient();
    CloudBlobContainer container = client.GetContainerReference(containerName);

    // ACT
    container.CreateIfNotExists();
    ICloudBlob blob = container.GetBlockBlobReference(filename);
    bool wasDeleted = blob.DeleteIfExists();

    // ASSERT
    Assert.That(wasDeleted, Is.EqualTo(true));
}
```

All the default functionality with blobs and containers creates them in a secure manner. All the previous tests that created containers actually created private containers. However, with different settings, you can also create public containers. Likewise, you can take a private container and modify it to be publicly available.

If you want to create a public container, it can be done with the container's SetPermissions method, in which you pass in a BlobContainerPermissions object preconfigured with PublicAccess enabled. You can also distribute storage access signatures to users to give temporary access to private containers or blobs via the GetSharedAccessSignature method on either the blob or container object. If you do this, be careful about the policy you set on the shared access signature.

Table and Queue storage

Table and Queue storage APIs are similar to Blob storage APIs at a high level. To work with tables and queues, you again start with the CloudStorageAccount object to access the obvious CreateCloudTableClient and CreateCloudQueueClient methods. Unlike blobs, queues and tables are always private and have no public access capabilities, so this simplifies the API a bit.

Queues are the simpler of the two in that there is no real searching functionality. Aside from the capability to get an approximate count of items in the queue, it's mostly just a push-and-pop mechanism to get messages into or off of the queue. The only interesting aspect of working with the queue API is what happens when you get a message. The act of getting a message does not actually perform the pop for that message. In fact, it causes that message to become invisible for a period of time (it defaults to one minute, but you can also specify it). During this period of time, you're guaranteed that nobody else will retrieve this message. If it takes you longer than this time period to process the message, you should periodically update the message to keep it hidden while you continue to process it. After you finish processing the message, you should then delete the message from the queue.

Tables are a little more interesting in that you can search the data and interact with it. The main way to fetch a single record is to execute a TableOperation. However, if you want to fetch many records, use a TableQuery. Following is a test showing the lifecycle of creating a table, adding a record, fetching that record, and then deleting that record. Notice the Record class it references and how it derives the TableEntity class. This enables you to quickly and easily manage the schema of the table as well as having .NET objects that can represent the data in the tables.

```
public class Record : TableEntity
{
    public Record() : this(DateTime.UtcNow.ToShortDateString(), Guid.NewGuid().
ToString())
    { }
    public Record(string partitionKey, string rowKey)
    {
        this.PartitionKey = partitionKey;
        this.RowKey = rowKey;
    }

    public string FirstName { get; set; }
    public string LastName { get; set; }
}

[TestCase("file")]
public void UploadTableFromConfig(string tableName)
{
    // ARRANGE
    CloudStorageAccount acct = CloudStorageAccount.Parse(ConfigurationManager.Connection
Strings["StorageConnection"].ConnectionString);
    CloudTableClient client = acct.CreateCloudTableClient();
    var table = client.GetTableReference(tableName);
    Record entity = new Record("1", "asdf"){FirstName = "Fred", LastName =
"Flintstone"};

    // ACT
    table.CreateIfNotExists(); // create table
    TableOperation insert = TableOperation.Insert(entity);
    table.Execute(insert); // insert record
    TableOperation fetch = TableOperation.Retrieve<Record>("1", "asdf");
    TableResult result = table.Execute(fetch); // fetch record
    TableOperation del = TableOperation.Delete(result.Result as Record);
    table.Execute(del); // delete record

    // ASSERT
    Assert.That(((Record)result.Result).FirstName, Is.EqualTo("Fred"));
}
```

Distribute data by using the Windows Azure Content Delivery Network (CDN)

The *Windows Azure Content Delivery Network (CDN)* is a way to cache Windows Azure blobs and static content. The idea is that you can use specific strategically placed nodes to maximize performance and bandwidth. Currently, you can specify node locations in the United

States, Europe, Asia, Australia, and South America. (When the number of missing nodes is greater than the number of nodes shown, it is probably a good sign that things are out of date and not worth going into.) In any case, the important takeaway is that you use CDN when performance is absolutely critical or when there are times you need to distribute the load placed on your resources (think Amazon on Cyber-Monday).

So the two primary benefits, according to MSDN, are these:

- Better performance and user experience for end users who are far from a content source, and are using applications in which many Internet trips are required to load content.
- Large distributed scale to better handle instantaneous high load, say at the start of an event such as a product launch.

This section is intentionally short because CDN-specific questions are rare in the exam because it's changed substantively (this is not an official statement from Microsoft). The UI, for instance, has changed regularly over the course of its existence, and screen shots I took from some previous writing on the subject were completely obsolete by the time I started writing this book.

CDN is an advanced feature, and although useful and necessary, it is something that's unquestionably going to evolve over the next few months and years. Focus on understanding what CDN is used for, the use cases, and a basic overview of it, and you should be prepared for the exam.

To use CDN, you must enable it on your storage account in the Windows Azure Management Portal. (At the time of the writing of this book, the CDN management tools are available only in the legacy version of the Windows Azure Management Portal.)

Next, understand how a request processed through CDN differs from a traditional request. Remember that a big selling point of the cloud is that you don't have to worry about many of the specifics about where things are stored (other than a URI). So when you make a request, the request is processed through the blob service or hosted service where your data is located. When the same thing is done with CDN enabled and configured, the request is transparently redirected to the closest endpoint to the location where the request was made. This minimizes the Internet trips mentioned previously. But there's a little more to it than this. All your data is not stored at the closest node, and you can't know in advance where the request will be made. So the only way to make sure that any given blob was at the closest node would be to have it on all nodes that were accessible. This is problematic because ,the first time a request is made, the item won't be at the closest node in many circumstances. In those cases, it is retrieved from the service and cached, actually causing a one-time performance hit prior to the performance gains. Subsequent requests to that node will use the cached item.

The only real nuance is staleness of the cached files. Anyone making any request after the first user gets a cached copy, so if the item changes frequently, it needs to be updated frequently. If it needs to be updated constantly (imagine a spreadsheet with stock quotes), the expiration needs to happen so frequently that it would offset any benefit from the caching.

Expiration is handled specifically by setting the *Time To Live (TTL)* value, which controls cache expiration.

How exactly do you make content available on CDN? First, it must exist in a public container. Second, the public container must be available for anonymous access. Again, only items that are publically available can be cached using CDN. This can be done through the Windows Azure Management Portal, but can also be facilitated through the API and the CloudBlobContainer class, for instance, in conjunction with the BlobContainerPermissions class. The CloubBlobContainer's Permissions property has a PublicAccess property of type BlobContainerPublicAccessType. The three values are Container, Off, and Blob. In this case, the container must be public, so BlobContainerPublicAccessType.Container should be used.

So after a container is set up, it's simply a matter of setting up the expiration policy and accessing the data. To access the content directly from the blob service, the URI is identical to what it was originally:

```
http://<storage account name>.blob.core.windows.net/<container>/<blob>
```

To access it directly through the Windows Azure CDN URL, use the following syntax, which is almost identical:

```
http://<storage account name>.vo.msecnd.net/<container>/<blob>
```

For the purposes of this exam, the hosted services component of CDN is pretty much identical other than the URIs. There are some other general things to keep in mind, though, when using CDN. Remember that you're using cached objects, so the more dynamic the object, the less likely it is a good candidate for caching. In fact, if it's too dynamic or the policy is too frequent in terms of expiration, it adversely affects performance. And that's part of the problem. What is too frequent? Every case will be different.

Manage Windows Azure Caching

Keep in mind that caching in Windows Azure is a completely different creature from the other types of caching discussed in this chapter. They are completely different. In a nutshell, it provides a caching layer to your Windows Azure applications. There are times when you benefit from caching with self-hosted applications after all, so why wouldn't the same hold true for Windows Azure-hosted applications? Moreover, CDN does cache data, but it's hardly the comprehensive caching solution needed for many much simpler scenarios.

Caching in the Windows Azure context has one additional advantage over traditional caching. In the Windows Azure context, it can greatly reduce costs associated with database transactions when using SQL databases. If you cache an item that is accessed 10,000 times, for instance, that's 10,000 fewer requests to the SQL database that are being made. Depending on the item, this could be quite substantial. I also point out that, if ignored or taken for granted, it could end up being a very costly mistake.

Caching in Windows Azure is known as *role-based caching*. That's because, as you might infer, it enables you to host caching within any Windows Azure role. Any given cache can be used by any of the other roles within the same cloud service deployment.

To facilitate this, there are two main mechanisms or topologies: dedicated topology and co-located topology. The names are the dead giveaway here, but in a dedicated topology, you define a role that is specifically designated to handle the caching. The Worker Role's available memory (all of it, if necessary) is dedicated to caching items in it and the necessary overhead to manage access to and from it.

In a co-located topology scenario, you assign specific thresholds, and only those thresholds or amounts under them can be used to facilitate the caching. If you had four Web Role instances, you could assign 25 percent to each of them.

Outside of these two mechanisms is an optional service known as *Windows Azure shared caching*. It's optional, and each separate service that employs caching is consumed as a managed service. These caches, unlike the role-based counterparts, do not exist in your own roles. They exist instead in a group of servers in a multitenant environment.

Handling exceptions by using retries (SQL Database)

Using a SQL Database in the cloud can be useful. You get a completely managed database without any prior configuration. It scales well and can be used from within your cloud or on-premise applications.

However, there is one thing different in the cloud than running a SQL database on-premise: latency. Because the physical distance between your database and application server is determined by the infrastructure in the datacenter, you get a higher latency than you would in an on-premise environment. Because of this, you will be more likely to experience timeouts when connecting to a database.

These types of errors are transient, meaning those errors will often go away after some time. For these types of errors, it makes sense to implement retry logic. This basically means that you inspect the error code that's returned from SQL Server, determine whether it's transient, wait for a set amount of time, and then try again to access the database.

The amount of time you wait before trying the action can be flexible. Maybe you want to retry three times, such as after one second, then two seconds, and then five seconds. Windows Azure offers the transient fault handling framework to help you with creating retry logic in your own applications. The following code shows an example of a retry strategy that retries on deadlocks and timeouts:

```
class MyRetryStrategy : ITransientErrorDetectionStrategy
{
    public bool IsTransient(Exception ex)
    {
        if (ex != null && ex is SqlException)
        {
            foreach (SqlError error in (ex as SqlException).Errors)
            {
                switch (error.Number)
                {
                    case 1205:
                        System.Diagnostics.Debug.WriteLine("SQL Error: Deadlock
condition. Retrying...");
                        return true;

                    case -2:
                        System.Diagnostics.Debug.WriteLine("SQL Error: Timeout expired.
Retrying...");
                        return true;
                }
            }
        }

        // For all others, do not retry.
        return false;
    }
}
```

You can use the retry strategy when executing a query from ADO.NET:

```
RetryPolicy retry = new RetryPolicy<MyRetryStrategy>(5, new TimeSpan(0, 0, 5));

using (SqlConnection connection = new SqlConnection(<connectionstring>))
{
    connection.OpenWithRetry(retry);

    SqlCommand command = new SqlCommand("<sql query>");
    command.Connection = connection;
    command.CommandTimeout = CommandTimeout;

    SqlDataReader reader = command..ExecuteReaderWithRetry(retry);

    while (reader.Read())
    {
        // process data
    }
}
```

When working on your application or service in Visual Studio you work with a lot of files. Some of those files contain code; others contain markup or configuration settings.

MORE INFO **TRANSIENT FAULT HANDLING FRAMEWORK**

For more information on the transient fault handling framework, see *http://social.technet.microsoft.com/wiki/contents/articles/4235.retry-logic-for-transient-failures-in-windows-azure-sql-database.aspx.*

Thought experiment

Choosing a Windows Azure strategy

In the following thought experiment, apply what you've learned about the "Implement data storage in Windows Azure" objective to design an appropriate Windows Azure strategy. You can find answers to these questions in the "Answers" section at the end of this chapter.

Your company is looking to move its core applications to the cloud. You have two primary applications: one that hosts video training files and another that provides advanced statistics about user interaction patterns. All the consumers of the application are currently clustered around three major metropolitan U.S. cities.

With these items in mind, answer the following questions:

1. **What storage options would you consider for each of the two scenarios?**
2. **Would one storage option suffice or would you want more than one?**
3. **Would this be a good candidate for the CDN? Why or why not?**

Objective summary

- Windows Azure offers a variety of storage options. Some of these are very similar to what on-premises servers utilize; others are very different.
- Local storage is available on most Windows Azure hosting offerings, including all cloud services and VMs.
- Blob storage is available for storing files in a durable file store. Blob storage can be empowered by the CDN to provide faster downloads and lower latencies for end users downloading files during heavy loads.
- Queue storage is similar to MSMQ and is the storage mechanism of choice for messages that need guaranteed delivery, even when the receiving system is down for hours at a time.
- Table storage is Windows Azure's NoSQL implementation. It allows for some very high-volume and high-speed inserts and accesses, much higher than what SQL databases allow if architected properly.
- Windows Azure offers you an additional caching framework that is very easy to use and can grow as your number of servers grows or can remain stable with dedicated instances.

Objective review

Answer the following question to test your knowledge of the information discussed in this objective. You can find the answer to this question and its corresponding explanation in the "Answers" section at the end of this chapter.

1. You are developing an ASP.NET application that hosts an online chat room. What Windows Azure Storage options should you consider for storing the contents of these chat rooms?
   - **A.** Table storage
   - **B.** Queue storage
   - **C.** Blob storage
   - **D.** CDN storage
2. Benefits of using Windows Azure Content Delivery Network include (choose all that apply)?
   - **A.** Increased performance for users geographically close to the content source.
   - **B.** Increased performance for users geographically far from a content source.
   - **C.** Large distributed scale to better handle instanteously increased loads.
   - **D.** Improved security.
3. Which of the following queries take advantage of the Content Delivery network?
   - **A.** *http://atlanta.vo.mysite.net/documents*
   - **B.** *http://myaccount.blob.core.windows.net/documents*
   - **C.** *http://myaccount.atlanta.blob.core.windows.net/documents*
   - **D.** *http://atlanta.mysite.net/documents*

Objective 1.5: Create and implement a WCF Data Services service

The discussion of WCF Data Services was short compared with the other technologies in Objective 1.1's coverage because it is discussed in depth in this objective. This section walks through creating and implementing a service and shows a few examples of how to work with it. Just keep in mind that, throughout the coverage, you specifically deal with WCF Data Services, WCF, the EF, and SQL Server. (Yes, there could be other data stores, but there aren't a lot of questions about MySql on the exam.)

This objective covers how to:

- Address resources
- Create a query
- Access payload formats (including JSON)
- Work with interceptors and service operators

Addressing resources

Visual Studio gives you ample tools for building and implementing WCF Data Services, but a basic walkthrough of how to do it is of little value for preparation of the exam. The approach in this section is to show you how to create the WCF Data Service and focus on generated content and what you need to know for the exam.

MSDN provides the following link, which has a start-to-finish walkthrough of creating a WCF Data Service (*http://msdn.microsoft.com/en-us/library/vstudio/cc668184.aspx*). Generally, you'll need to do the following:

1. Create an ASP.NET application (this serves as the host for the Data Service).
2. Use the EF tools to build an EntityModel.
3. Add a WCF Data Service to the ASP.NET application.
4. Specify a type for the Service's definition (which is the name of the model container created in your EF project).
5. Enable access to the Data Service. This is accomplished by explicitly setting specific properties of the DataServiceConfiguration (SetEntitySetAccessRule, SetServiceOperationAccessRule and DataServiceBehavior).

When you add a WCF Data Service to an ASP.NET application (and you can certainly use other hosting mechanisms and hand-code all this if you're so inclined; it's just a lot more cumbersome and error-prone), you get a class definition that looks something like the following:

```
public class ExamSampleService : DataService<FILL IN MODEL NAME NERE>
{}
```

First, make sure that you understand that WCF Data Services inherit from the System.Data.Services.DataService base class. The constructor takes in a generic type that is indicated by the template class and is absolutely instrumental to the way everything operates.

The next thing you can see in the generated code is the InitializeService method, which takes in a parameter of type DataServiceConfiguration. The generated code is marked with a TODO comment and some commented-out code that is used as a visual tool to help you get started. It looks like this:

```
// This method is called only once to initialize service-wide policies.
public static void InitializeService(DataServiceConfiguration config)
{
 // TODO: set rules to indicate which entity sets and service operations are visible,
updatable, etc.
 // Examples:
 // config.SetEntitySetAccessRule("MyEntityset", EntitySetRights.AllRead);
 // config.SetServiceOperationAccessRule("MyServiceOperation", ServiceOperationRights.
All);
 config.DataServiceBehavior.MaxProtocolVersion = DataServiceProtocolVersion.V2;
}
```

There are several properties and methods defined in the DataServiceConfiguration class, but the important ones to be familiar with are the SetEntitySetAccessRule method, the SetServiceOperationAccessRule method, and the DataServiceBehavior.MaxProtocolVersion property.

SetEntityAccessRule

This method takes two parameters and is very easy to understand and use. The first parameter is a string that names the entity set that the next parameter applies to. The next parameter is the Rights parameter, which is part of the EntitySetRights enumeration. The one noteworthy thing about this enumeration is that it is decorated with the Flags attribute and intended to be used that way. So, for example, when the access rules for the Courses entity set and give it AllRead and WriteMerge permissions, the following definition is used:

```
config.SetEntitySetAccessRule("Courses", EntitySetRights.AllRead | EntitySetRights.
WriteMerge);
```

Because it's likely to come up in one form or another, walk through the EntitySetRights enumeration's possible values. The names are intuitive, but it's likely that you'll see something in a question stub that might tip you off to the necessity of one or more of them. Table 1-8 shows each member and behavior of the EntitySetRights enumeration.

TABLE 1-8 EntitySetRights

Member name	Behavior
None	All rights to the data are explicitly revoked.
ReadSingle	Single data items can be read.
ReadMultiple	Entire sets of data can be read.
WriteAppend	New items of this type can be added to data sets.
WriteReplace	Data can be updated or replaced.
WriteDelete	Data can be deleted.
WriteMerge	Data can be merged.
AllRead	Across-the-board access to read data of this type.
AllWrite	Across-the-board access to write data of this type.
All	All Creation, Read, Update and Delete operations can be performed.

It should go without saying, but just for the sake of being clear, none of these override permissions are set by the DBA or defined at the database level. You can have the All permission, for instance, but if the DBA revokes your access to the database or that object, you won't be able to access it just because it's specified here.

What might not seem quite as obvious, though, is that EntitySetRight values can be overruled by ServiceOperationRights. As you'll see, ServiceOperationRights are intended to be used as flags, so whatever other values are specified, the OverrideEntitySetRights value can be set, too. When there's a conflict, EntitySetRights lose to both the database's permission and the ServiceOperationRights.

SetServiceOperationAccessRule

This method is commented out, but it is part of the TODO section, as you saw with EntitySetRights. It pertains to any given operation name, and it too defines the permissions that should be applied to the operation through the ServiceOperationRights enumeration (which is also decorated with the flags attributed and is meant to be used as such).

```
config.SetServiceOperationAccessRule("OperationName", ServiceOperationRights.All);
```

Table 1-9 describes this enumeration in detail, and again it's worth a look so you can recognize values when you see them if they appear on the exam.

TABLE 1-9 ServiceOperationRights

Member name	Behavior
None	No authorization to access the operation is granted.
ReadSingle	One single data item can be read using this operation.
ReadMultiple	Multiple data items can be read using this operation.
AllRead	Single and multiple data item reads are allowed.
All	All rights are granted to the service operation.
OverrideEntitySetRights	Overrides EntitySetRights that are explicitly defined in the Data Service.

DataServiceBehavior.MaxProtocolVersion

This value affects the service only to the extent that it will be used with OData (this feeble attempt at humor is actually intentional; it's there to drive home how important this seemingly simple and mundane property is).

Table 1-10 shows the allowable values of this enumeration (which is *not* decorated with the Flags attribute, as you probably guessed).

TABLE 1-10 DataServiceProtocolVersion

Member name	Behavior
V1	Supports version 1.0 of the OData Protocol
V2	Supports version 2.0 of the OData Protocol

Obviously, I'm not saying you need to know nothing else, but for the exam, make sure that you understand fully what the constructor for a WCF Data Service looks like, what base class it inherits from, how to specify the generic type parameter in the base class' constructor, and the details discussed previously.

Creating a query

Sometimes it's a little awkward to discuss things in different sections. This is one of those cases. QueryInterceptors and ChangeInterceptors would probably fit pretty well in this discussion, but I will defer coverage until the end because they deserve their own section, that and the fact that you need to understand how to query data before some of the coverage for those features will make sense.

OData Support is one of the main reasons to use WCF Data Services, and it is URI addressable. You can query data and do quite a bit with it by changing and manipulating URIs. For each example, assume that you have a WCF Service defined and it is named the ExamPrepService. It is based on an Entity Model that contains topics, questions, and answers as entities.

If you want to get all the topics from the service, use the following query:

```
http://servicehost/ExamPrepService.svc/Topics
```

Use Service/EntitySetName to get all the entities. Assume that you want to do the same thing, but instead of wanting all Topic items returned, you want just the Topic named "First" returned:

```
http://servicehost/ExamPrepService.svc/Topics('First')
```

In this case, it is Service/EntitySetName to get the entity, ('KeyValue') to restrict results to the item that has a key matching items in the parentheses.

Assume that this time you want to do the same as the previous example, but you need only the Description property returned. Here's how to do it:

```
http://servicehost/ExamPrepService.svc/Topics('First')/Description
```

So to return just the specified property, add another slash to the previous query and add the Property name to it.

At any time, if you want just the primitive type, not the corresponding XML, you can accomplish it by appending $value at the end of your URI (query). Using the previous example, returning just the primitive value string, just use the same query, with /$value appended on the end:

```
http://servicehost/ExamPrepService.svc/Topics('First')/Description/$value
```

As long as there are relationships defined in the Entity Model, you can use a few different semantics to return related entities. Each of the following is supported:

- Parent entity—Specific child entity
- Parent entity—All child entities
- Set of related entities

Change things up and assume (just for review) that you want to return the Question entity with a key value of "1" and you want just the QuestionText property:

```
http://servicehost/ExamPrepService.svc/Questions('1')/QuestionText
```

That would give you the Question text, but what if you wanted the Answers to it? You would use:

```
http://servicehost/ExamPrepService.svc/Questions('1')/Answers
```

You can work it the other way around, too, of course. If you want the Question that corresponded to an Answer entity with a key of ('1:4'):

```
http://servicehost/ExamPrepService.svc/Answers('1:4')/Question
```

You can technically "filter" data using the previous semantics, but filtering can be applied using the filter keyword, and in many cases needs to be. In the previous example, you returned questions that had a key value of '1,' which had the effect of filtering it to a SQL statement that might look like the following:

```
SELECT Field1, Field2, Field3 etc from Questions WHERE KEY = '1'
```

All the previous semantics filter only according to key values. Sure, keys are good (and frequent) properties to run queries off of, but there are times when you need to use other values and combinations of values. If you need to restrict based off of anything other than key, you'll need to use the filter keyword.

EXAM TIP

Most developers and even computer science students who are not just starting their programs are familiar with SQL. SQL is easy to learn, and it's so common that most people have come across it. Although I can run through every single query and feature available through OData and toying with the URIs, understanding how different URIs translate to SQL might be helpful to drive home your understanding. I've seen quite a few cases in which people have missed questions on this exam that they unquestionably knew the SQL for, but got confused on the OData portion. If you create a service, turn on Profiler in SQL Server and watch the results as you change different URIs; you'll see exactly what it's being translated to behind the scenes. Watching what's going on in Profiler is a very helpful learning tool in many cases, not just this one. It does add a slight resource load to the SQL Server, so clear it with your DBA or use an instance that you know isn't being used by anyone for anything important at the time. I highly recommend that you do a quick runthrough, watching what happens in Profiler just to drive the point home if you have any extra time to do so.

If you want to return all the topics greater than Topic 1, though, using the previous semantics wouldn't work. What if you want to return just topic items that had the phrase "Data Service" in their text? Because their Text property isn't the key, you couldn't use what you have done so far. You don't have a way to append X number of additional keys into the one parenthesis, and at this point, you're not even sure whether that will work (it doesn't) or even necessarily know how many different topics there are. Who knows, that might be what you want to find out. So that's where filters come in. If you need a range of values, if you need to restrict data based on something that isn't a key field, or if you have multiple conditions, you need to start using the $filter value.

To tailor the behavior of queries, you need to take advantage of the various OData query options. $filter is one of the most common.

MORE INFO **ODATA FILTER OPERATORS**

OData.org (*http://www.odata.org/documentation/odata-v2-documentation/uri-conventions/*) provides a full list of the OData URL conventions and how to use them. Refer to it if you have any questions about specific usage.

Table 1-11 shows a list of each of the query options available for OData through WCF Data Services.

TABLE 1-11 Query options

Member name	Behavior
$orderby	Defines a sort order to be used. You can use one or more properties that need to be separated by commas if more than one will be used: *http://servicehost/ExamPrepService.svc/Questions?$orderby=Id,Description*
$top	The number of entities to return in the feed. Corresponds to the TOP function in SQL: *http://servicehost/ExamPrepService.svc/Questions?$top=5*
$skip	Indicates how many records should be ignored before starting to return values. Assume that there are 30 topics in the previous data set, and you want to skip the first 10 and return the next 10 after that. The following would do it: *http://servicehost/ExamPrepService.svc/Questions?$skip=10&$top=5*
$filter	Specifies a condition or conditions to filter on: *http://servicehost/ExamPrepService.svc/Questions?$filter=Id gt 5*
$expand	Indicates which related entities are returned by the query. They will be included either as a feed or as an entry inline return with the query: *http://servicehost/ExamPrepService.svc/Questions?$expand=Answers*
$select	By default, all properties of an entity are returned in the feed. This is the equivalent of SELECT * in SQL. By using $select, you can indicate a comma separated list to just return the fields you want back: *http://servicehost/ExamPrepService.svc/Questions&$select=Id,Text,Description,Author*
$inlinecount	Returns the number of entries returned (in the <count> element). If the following collection had 30 total values, the feed would have a <count> element indicating 30: *http://servicehost/ExamPrepService.svc/Questions?$inlinecount=allpages*

In addition to making requests through a URI, you can execute queries through code imperatively. To facilitate this, the DataServiceQuery class is provided.

If you want to use the DataServiceQuery approach to get the same functionality, you build your service just as you already have, and you set up a DataServiceQuery instance and use the AddQueryOptions method. A quick sample should suffice:

```
String ServiceUri = "http://servicehost/ExamPrepService.svc";
ExamServiceContext ExamContext = new ExamServiceContext(new Uri(ServiceUri);
DataServiceQuery<Questions> = ExamContext.Question
                              .AddQueryOptions("$filter", "id gt 5")
                              .AddQueryOptions("$expand", "Answers");
```

You can, of course, accomplish the same thing using straight LINQ/EF syntax, but because that approach would have nothing specific to WCF Data Services to it, it might appear on some other exam or portion of the exam (but probably not appear here).

EXAM TIP

In the exam, you'll probably be presented with some definition of an Entity Model (even if it's just a small piece of it) and then ,told that you need to "create a query that... ." You'll then be presented with a list of possible queries that would return the desired results. Understanding the items in Table 1-12 and how they affect the resulting query is indispensable for answering such questions correctly. $orderby, $select, $top, and $inlinecount are obvious in what they do and how they are used, and there's not much to them. That leaves $filter, $skip, and $expand to study.

Accessing payload formats

There is support for both Atom and JSON formats. They are easy to use and are intuitive; you use the $format option with one of two values: $format=atom or $format=json. If you decide instead to access it via the WebClient or by specifying it through the request header, it works the same way, with just a small change: You need to append "application/" to the headers. To use JSON, you simply need to specify "application/json" or "application/atom+xml."

The issue of JSON and Atom as payload formats appears extensively in other portions of the exam, but in terms of the Data Service component, there's not much more to know than what was mentioned previously.

Working with interceptors and service operators

The WCF Data Services infrastructure enables you to intercept requests and provide custom logic to any given operation. Interceptors, as the name implies, are the mechanism you can use to accomplish this. When a request is made, it can be intercepted, and additional custom logic can be applied to the operation. Common use cases for interception include validation of inbound messages and changing the scope of a request.

To facilitate interception, you use the Interceptor type name, passing in the corresponding parameters, and decorate the method with it.

There are two basic types of interceptors you should be familiar with: ChangeInterceptors and QueryInterceptors. As the names imply, they have distinct usages depending on what you are looking for. ChangeInterceptors are used for NON-Query operations; QueryInterceptors are used for Query operations.

ChangeInterceptors are used for all nonquery operations and have no return type (void in C#, Nothing in VB.NET). They must accept two parameters:

- **Type** A parameter of type that corresponds to the entity type associated with the entity set.
- **UpdateOperations** When the operation is invoked, this value references the request that you want to perform.

The definition for the attribute on a method handling OnChangeTopics looks like the following:

```
[ChangeInterceptor("Topics")]

public void OnChangeTopics(Topic topic, UpdateOperations operations)
```

According to MSDN, QueryInterceptor items must meet the following conditions.

- Entity set authorization and validation is handled by methods decorated with the QueryInterceptor attribute.
- Entity set access control and validation is enabled through query operations by using composition. To accomplish this, the following conditions must be met:
  - The method must have public scope.
  - It must be decorated with the QueryInterceptor attribute.
  - The QueryInterceptor must take the name of an entity set as a parameter.
  - The method must not take any parameters.
  - The method must return an expression of type Expression<Func<T, Boolean>> that serves as the filter for the entity set.

So the signature for a method implementing a QueryInterceptor looks like the following:

```
[QueryInterceptor("Topics")]
public Expression<Func<Topic, bool>> FilterTopics(){}
```

The tipoff for which to use and when to use it is determined by what it is used for. It is denoted by the signature of the method. If you see <Func<SomeEntity, bool>> Whatever(), you can tell immediately that it's a QueryInterceptor. If you see anything else, it's a ChangeInterceptor.

Similarly, you can tell by the behavior which is which and which is being asked for. DeleteTopics would be a ChangeInterceptor question; a GetAdvancedTopics method asking about filtering Topics entity would be a QueryInterceptor question.

Thought experiment

Querying data services

In the following thought experiment, apply what you've learned about the "Create and implement WCF Data Services" objective to design a transport server infrastructure. You can find answers to these questions in the "Answers" section at the end of this chapter.

Your company has an application that currently is consumed by in-house applications. There is a need to make it accessible to outside parties that use this in-house application but extend it. As such, the capability to consume and manipulate the data is critical. Enabling users to accomplish this through URI-based semantics seems like the optimal solution.

With these facts in mind, answer the following questions:

1. **Would it be possible to provide access through URI-based semantics using Data Services?**
2. **What operations are easily supported?**

Objective summary

- WCF Data Services provide easily accessible data operations through OData.
- WCF Data Services can make use of both JSON and Atom.
- The SetEntitySetAccessRule controls the how an entity can be queried.
- The EntitySetRights enumeration is created with the Flags attribute and is intended to be used as such. In many or most cases, you'll specify more than one value for such access rules.
- Queries can be performed using URIs, and tremendous functional enhancements can be implemented by small changes.
- QueryOperations are denoted with the $ at the beginning of an operation and can control a wide variety of functions, including the number of items returned to specifying the TOP x of the resultset.

Objective review

Answer the following questions to test your knowledge of the information discussed in this objective. You can find the answers to these questions and their corresponding explanations in the "Answers" section at the end of this chapter.

1. You need to create an interceptor that runs each time you try to return Topic items. Which items should you use? (Choose all that apply.)
   - **A.** ChangeInterceptor
   - **B.** QueryInterceptor
   - **C.** DataServiceProtocolVersion
   - **D.** SetEntitySetAccessRule
2. You need to return all questions that have an Id greater than 5. Which query should you use?
   - **A.** *http://servicehost/ExamPrepService.svc/Questions?$filter=Id gt 5*
   - **B.** *http://servicehost/ExamPrepService.svc/Questions?$filter(ID> 5)*
   - **C.** *http://servicehost/ExamPrepService.svc/Questions(>5)*
   - **D.** *http://servicehost/ExamPrepService.svc/Questions?$select(Id)* gt 5
3. Assume you have the following C# code snippet.

```
var selectedQuestions = from q in context.Questions
                     where q.QuestionNumber > 30
                     orderby q.QuestionId descending
                     select q;
```

   Which of the following URI queries is the equivalent?
   - **A.** *http://Service/Question.svc/Questions?Orderby=QuestionId&?$QuestionNumber (gt 30)*
   - **B.** *http://Service/Question.svc/Questions?Orderby=QuestionId&?$QuestionNumber gt 30*
   - **C.** *http://Service/Question.svc/Questions?Orderby=QuestionId&?filte r=(QuestionNumber > 30)*
   - **D.** *http://Service/Question.svc/Questions?Orderby=QuestionId&?filte r=QuestionNumber gt 30*

Objective 1.6: Manipulate XML data structures

Ever since it showed up on the scene, XML has made a huge impact. Today, virtually every application makes use of XML in some way or other. In some cases, it is used as a serialization format. In others, it is used to store configuration data in a way that doesn't necessitate registry access and the permissions such access requires. It is also used as a basis for Web Services and as a file format (as a matter of fact, it's the underlying file format for this document). These are just a few areas of what XML is used for. XML is the answer to so many problems that plagued the development world that it's one of the few technologies that not only lived up to the hype that surrounded it (and there was plenty) but it also completely exceeded the hype by a huge margin.

This objective covers how to:

- Read, filter, create, and modify XML structures
- Manipulate XML data by using XMLReader, XMLWriter, XMLDocument, XPath, and LINQ-to-XML
- Advanced XML manipulation

Reading, filtering, creating, and modifying XML structures

The first component of an XML Document is typically known as the *XML declaration*. The XML declaration isn't a required component, but you will typically see it in an XML Document. The two things it almost always includes are the XML version and the encoding. A typical declaration looks like the following:

```
<?xml version="1.0" encoding="utf-8" ?>
```

You need to understand the concept of "well-formedness" and validating XML. To be well-formed, the following conditions need to be met:

- There should be one and only one root element.
- Elements that are opened must be closed and must be closed in the order they were opened.
- Any element referenced in the document must also be well-formed.

The core structures of XML are *elements* and *attributes*. Elements are structures that represent a component of data. They are delineated by less-than and greater-than signs at the beginning and end of a string. So an element named FirstName looks like this:

```
<FirstName>
```

Each element must be closed, which is indicated by slash characters at the beginning of the element:

```
</FirstName>
```

To define an element named FirstName with the value "Fred" in it, this is what it would look like:

```
<FirstName>Fred</FirstName>
```

If an element has no data value, you can represent it in one of two ways:

- An opening element followed by a closing element with no value in between them:

  ```
  <FirstName></FirstName>
  ```

- An opening element with a slash at the end of string instead of the beginning:

  ```
  <FirstName/>
  ```

Attributes differ from elements in both syntax and nature. For instance, you might have the following structure that describes a "Name":

```
<?xml version="1.0" encoding="utf-8" ?>
<Name>
  <FirstName>John</FirstName>
  <MiddleInitial>Q</MiddleInitial>
  <LastName>Public</LastName>
</Name>
```

Name in this case is its own element, and FirstName, MiddleInitial, and LastName are their own elements, but have context and meaning only within a Name element. You could do the same thing with attributes, although they are necessarily part of the element to which they belong:

```
<Name FirstName="John" MiddleInitial="Q" LastName="Public"></Name>
```

If you tried to consume this data, the way you access it would differ, but the end result would be that you'd retrieve information about a Name, and the data would be identical. Which is better? Which one should you use? There's no correct answer to either question. It depends on style and personal preference. The Internet is full of passionate debate about this subject, which is testimony to the fact that there is not a definitive right or wrong answer for every case.

Manipulating elements and attributes are the crux of what you'll encounter on the exam, but for the sake of being thorough, there are two other items you should be familiar with: comments and namespaces. I can't say that you'll never need to concern yourself with retrieving comment information, but it's not something you come across very often, and I've never had to do it (and I've done a lot of XML parsing). The main thing you need to know is simply how to identify comments so you can distinguish them from other XML elements. You delineate a comment with the following character sequence:

```
<!-- Then you end it with the following:-->
```

So a full comment looks like this:

```
<!-- this is a comment about the Name element. Blah blah blah-->
```

Namespaces are a little more involved. Assume that you want to use an element name—something common. If namespaces didn't exist, it would mean that, after an element name was used, it couldn't be used for anything else. You can imagine how much difficulty this would cause when you're dealing with different vendors, all adding to an existing snippet of XML. This would be particularly problematic even if you didn't have different vendors but had a case in which different XML fragments were used. If you are familiar with DLL Hell, this is its evil cousin.

So namespaces were added to the spec. You can define namespaces in the root node or in the element node. In either case, they are delineated with the following syntax:

```
xmlns:Prefix="SomeValueUsuallyACompanyUrl"
```

The xmlns portion is what denotes a namespace declaration (*xml NameSpace* is abbreviated to xmlns). You then specify what prefix you want to use (Prefix in the previous example). Then you use an equal sign and give it a unique value. You could use any value you know to be unique, but using your own company URL is usually a good practice, coupled with something specific about the namespace. If you use namespaces only in the context of your company, you can use a slash and some other name that you know to be unique. If you don't, you'll have a collision that will make things confusing. So using this approach, here's what the document definition would look like along with an example of each being used in an element:

```
<DocumentCore xmlns:johnco="http://www.yourcompany.com/Companies" xmlns:billco="http://
www.mycompany.com/Customers">
  <johnco:Name>
    <johnco:Company>JohnCo</johnco:Company>
  </johnco:Name>
  <billco:Name>
    <billco:FirstName>John</billco:FirstName>
    <billco:MiddleInitial>Q</billco:MiddleInitial>
    <billco:LastName>Public</billco:LastName>
  </billco:Name>
</DocumentCore>
```

The previous example includes two different vendors, BillCo and JohnCo, that each happened to use an element named Name. Once you define a namespace, you simply prefix the element with the namespace and a colon, and then include the element name, as indicated previously.

You can also define namespaces at the element level instead. This is the same principle with just a slightly different definition. In general, it's more concise to define the namespaces globally if you have repeated instances of an element. Think of several Name instances of both the johnco: and the billco: Name element. Defining it inline each time would be repetitive, inefficient, and a lot harder to read. The following shows how to define a namespace inline:

```
<DocumentCore>
  <johnco:Name xmlns:johnco="http://www.yourcompany.com/Companies">
    <johnco:Company>JohnCo</johnco:Company>
  </johnco:Name>
  <billco:Name xmlns:billco="http://www.mycompany.com/Customers">
    <billco:FirstName>John</billco:FirstName>
    <billco:MiddleInitial>Q</billco:MiddleInitial>
    <billco:LastName>Public</billco:LastName>
  </billco:Name>
</DocumentCore>
```

The pros and cons of using each approach is beyond the scope of this discussion and not relevant for the test. You simply need to know that both forms of the syntax are valid and get you to the same place.

Manipulating XML data

The previous items are all the primary classes you can use to manipulate XML data outside of the LINQ namespace. They belong to the System.Xml namespace and all work essentially the same way. They have all been around for a while as far as the .NET Framework is concerned, and it's doubtful they'll comprise much of the exam as far as XML manipulation goes. They are important, but they have been around since version 1 of the Framework, and you're much more likely to encounter questions focused on LINQ. A basic familiarity with them, knowledge of their existence, and a basic understanding of how they work should more than suffice for the purposes of the exam.

XmlWriter class

The XmlWriter class can be used to write out XmlDocuments. It's intuitive to use and needs little explanation. The steps are as follows:

- Create a new instance of the XmlWriter Class. This is accomplished by calling the static Create method of the XmlWriter class (and for convenience, passing in a file name as a parameter).
- Call the WriteStartDocument method to create the initial document.
- Call the WriteStartElement, passing in a string name for the element name for the root element.
- Use the WriteStartElement again to create an instance of each child element of the root element you just created in the previous step.
- Use the WriteElementString method passing in the element name and value as parameters.
- Call the WriteEndElement method to close each element you created.
- Call the WriteEndElement method to close the root element.
- Call the WriteEndDocument method to close the document you created initially.

There are several other methods you can use, such as WriteComment, WriteAttributes, or WriteDocType. Additionally, if you opt to use Asynchronous methodology, you can call the corresponding Async methods that bear the same names, but have Async at the end of them.

NOTE **SAMPLE CODE IS FOCUSED ON BEING READABLE**

I intentionally left out items such as overloading base class methods and some other things I'd include in production code for the purposes of readability. So the class definition is hardly an example of an ideal sample of production code. In the same respect, the exam has to take readability into account, so it's likely to follow similar conventions.

Assume that you have the following class definition for Customer:

```
public class Customer
{
    public Customer() { }
    public Customer(String firstName, String middleInitial, String lastName)
    {
        FirstName = firstName;
        MiddleInitial = middleInitial;
        LastName = lastName;
    }
    public String FirstName { get; set; }
    public String MiddleInitial { get; set; }
    public String LastName { get; set; }
}
```

The following shows code based on the class definition and follows the steps outlined in the previous list:

```
public static class XmlWriterSample
{
    public static void WriteCustomers()
    {
        String fileName = "Customers.xml";
        List<Customer> customerList = new List<Customer>();
        Customer johnPublic = new Customer("John", "Q", "Public");
        Customer billRyan = new Customer("Bill", "G", "Ryan");
        Customer billGates = new Customer("William", "G", "Gates");
        customerList.Add(johnPublic);
        customerList.Add(billRyan);
        customerList.Add(billGates);

        using (XmlWriter writerInstance = XmlWriter.Create(fileName))
        {
            writerInstance.WriteStartDocument();
            writerInstance.WriteStartElement("Customers");

            foreach (Customer customerInstance in customerList)
            {
                writerInstance.WriteStartElement("Customer");
                writerInstance.WriteElementString("FirstName", customerInstance.
FirstName);
                writerInstance.WriteElementString("MiddleInitial", customerInstance.
MiddleInitial);
                writerInstance.WriteElementString("LastName", customerInstance.
LastName);
                writerInstance.WriteEndElement();
            }
            writerInstance.WriteEndElement();
            writerInstance.WriteEndDocument();
        }
    }
}
```

This code produces the following output:

```
<?xml version="1.0" encoding="UTF-8"?>
<Customers>
 <Customer>
  <FirstName>John</FirstName>
  <MiddleInitial>Q</MiddleInitial>
  <LastName>Public</LastName>
</Customer>-<Customer>
  <FirstName>Bill</FirstName>
  <MiddleInitial>G</MiddleInitial>
  <LastName>Ryan</LastName>
</Customer>-<Customer>
  <FirstName>William</FirstName>
  <MiddleInitial>G</MiddleInitial>
  <LastName>Gates</LastName>
 </Customer>
</Customers>
```

XmlReader class

The XmlReader is the counterpart to the XmlWriter, and it's equally simple to use. Although there are several different cases you can check for (attributes, comments, namespace declarations, and so on), in its simplest form, you simply do the following:

- Instantiate a new XmlReader instance passing in the file name of the XML file you want to read.
- Create a while loop using the Read method.
- While it iterates the elements, check for whatever you want to check looking at the XmlNodeType enumeration.

The following method iterates the document created in the previous section and outputs it to the console window:

```
public static void ReadCustomers()
{
    String fileName = "Customers.xml";
    XmlTextReader reader = new XmlTextReader(fileName);
    while (reader.Read())
    {
        switch (reader.NodeType)
        {
            case XmlNodeType.Element: // The node is an element.
                Console.Write("<" + reader.Name);
                Console.WriteLine(">");
                break;
            case XmlNodeType.Text: //Display the text in each element.
                Console.WriteLine(reader.Value);
                break;
            case XmlNodeType.EndElement: //Display the end of the element.
                Console.Write("</" + reader.Name);
                Console.WriteLine(">");
                break;
        }
    }
}
```

There’s no need to go through each item available in the XmlNodeType enumeration, but you can become familiar with the available items on MSDN: *http://msdn.microsoft.com/en-us/library/system.xml.xmlnodetype.aspx*.

XmlDocument class

The XmlDocument class is the parent of the others in some ways, but it’s even easier to use. You typically do the following:

- Instantiate a new XmlDocument class.
- Call the Load method pointing to a file or one of the many overloaded items.
- Extract the list of nodes.
- Iterate.

The following code shows how to walk through a Node collection and extracts the InnerText property (which is the value contained in the nodes). Although there are other properties you can take advantage of, this chapter is about data and working with it:

```
String fileName = "Customers.xml";
XmlDocument documentInstance = new XmlDocument();
documentInstance.Load(fileName);
XmlNodeList currentNodes = documentInstance.DocumentElement.ChildNodes;
foreach (XmlNode myNode in currentNodes)
{
    Console.WriteLine(myNode.InnerText);
}
```

Writing data using the XmlDocument class works intuitively. There’s a CreateElement method that accepts a string as a parameter. This method can be called on the document itself (the first of which creates the root node) or any given element. So creating an initial document and then adding a root node named Customers that contains one element named Customer is created like this:

```
XmlDocument documentInstance = new XmlDocument();
XmlElement customers = documentInstance.CreateElement("Customers");
XmlElement customer = documentInstance.CreateElement("Customer");
```

In order to make this work right, you must remember the rules of well-formedness (and these in particular):

- Any tag that’s opened must be closed (explicitly or with an close shortcut for an empty element, i.e., <FirstName/>.
- Any tag that’s opened must be closed in a Last Opened First Closed manner. <Custo mers><Customer>SomeCustomer</Customer></Customers> is valid; <Customers> <Customer>SomeCustomer</Customers></Customer> is not.

To that end, in the previous code, the XmlElement named Customers should be the last of the group to have a corresponding AppendChild method called on it, followed only by the AppendChild being called on the document itself.

One more thing needs to be mentioned here. The CreateElement method simply creates the element; it does nothing else. So if you want to create an element named FirstName and then add a value of John to it, use the following syntax:

```
XmlElement FirstNameJohn = DocumentInstance.CreateElement("FirstName");
FirstNameJohn.InnerText = "John";
```

The following segment shows the process, from start to finish, of creating the output specified after it:

Code

```
XmlDocument documentInstance = new XmlDocument();
XmlElement customers = documentInstance.CreateElement("Customers");
XmlElement customer = documentInstance.CreateElement("Customer");
XmlElement firstNameJohn = documentInstance.CreateElement("FirstName");
XmlElement middleInitialQ = documentInstance.CreateElement("MiddleInitial");
XmlElement lastNamePublic = documentInstance.CreateElement("LastName");
firstNameJohn.InnerText = "John";
middleInitialQ.InnerText = "Q";
lastNamePublic.InnerText = "Public";
customer.AppendChild(firstNameJohn);
customer.AppendChild(middleInitialQ);
customer.AppendChild(lastNamePublic);
customers.AppendChild(customer);
documentInstance.AppendChild(customers);
```

Output

```
<Customers>
   <Customer>
     <FirstName>John</FirstName>
     <MiddleInitial>Q</MiddleInitial>
     <LastName>Public</LastName>
   </Customer>
</Customers>
```

If you wanted to add additional Customer elements, you'd follow the same style, appending them to the corresponding parent element in the same manner as you did here.

For attributes, there's a SetAttribute method that accepts two strings as parameters and can be called on any given element. The first string is the attribute name; the second is the attribute value. Using the example, you can attain the same goal you accomplished earlier by using the XmlDocument class, as shown in the following:

Code

```
String fileName = "CustomersPartial2.xml";
XmlDocument documentInstance = new XmlDocument();
XmlElement customers = documentInstance.CreateElement("Customers");
XmlElement customer = documentInstance.CreateElement("Customer");
customer.SetAttribute("FirstNameJohn", "John");
customer.SetAttribute("MiddleInitialQ", "Q");
customer.SetAttribute("LastNamePublic", "Public");
customers.AppendChild(customer);
documentInstance.AppendChild(customers);
documentInstance.Save(fileName);
```

Output

```
<Customers>
<Customer LastNamePublic="Public" MiddleInitialQ="Q" FirstNameJohn="John"/>
</Customers>
```

XPath

One feature of navigating through a document is *XPath*, a kind of query language for XML documents. XPath stands for XML Path Language. It's a language that is specifically designed for addressing parts of an XML document.

XmlDocument implements *IXPathNavigable* so you can retrieve an XPathNavigator object from it. The XPathNavigator offers an easy way to navigate through an XML file. You can use methods similar to those on an XmlDocument to move from one node to another or you can use an XPath query. This allows you to select elements or attributes with certain values.

Let's say you are working with the following XML:

```
<?xml version="1.0" encoding="utf-8" ?>
<People>
  <Person firstName="John" lastName="Doe">
    <ContactDetals>
      <EmailAddress>john@unknown.com</EmailAddress>
    </ContactDetals>
  </Person>
  <Person firstName="Jane" lastName="Doe">
    <ContactDetals>
      <EmailAddress>jane@unknown.com</EmailAddress>
      <PhoneNunmber>001122334455</PhoneNunmber>
    </ContactDetals>
  </Person>
</People>
```

You can now use an XPath query to select a *Person* by name:

```
XmlDocument doc = new XmlDocument();
doc.LoadXml(xml);

XPathNavigator nav = doc.CreateNavigator();
string query = "//People/Person[@firstName='Jane']";
XPathNodeIterator iterator = nav.Select(query);

Console.WriteLine(iterator.Count); // Displays 1

while(iterator.MoveNext())
{
    string firstName = iterator.Current.GetAttribute("firstName","");
    string lastName = iterator.Current.GetAttribute("lastName","");
    Console.WriteLine("Name: {0} {1}", firstName, lastName);
}
```

This query retrieves all people with a first name of Jane. Because of the hierarchical structure of XML, an XPath query can help you when you're trying to retrieve data.

MORE INFO **XPATH LANGUAGE**

For a complete overview of the XPath language, see *http://www.w3.org/TR/xpath/*.

LINQ-to-XML

LINQ will likely be featured prominently in the exam. Entire books are written on LINQ and how to use it; this coverage is not intended to be a comprehensive discussion of LINQ (or anything even close to it). It does, however, cover the elements (no pun intended) that you're likely to encounter as you take the exam.

There's one point that can't be emphasized enough. The more you know about the technology, the more likely you are to be able to rule out incorrect answers. I have not only taken several certification exams and participated in every aspect of the process more times than I can count, but I have also been an item writer. Trust me; it's not an easy task. I won't go into all the details about it, but you almost always have to rely on subtle differences to come up with valid questions that adequately differentiate them from each other. The more you know about the technology, the more likely you are to pick up on something that just doesn't look right or that you know can't be the case. In most instances, that just increases the probability of guessing the right answer. On the visual drag-and-drop questions, having such knowledge can enable you to use the process of elimination, which can greatly increase your chances of getting the question right. LINQ semantics feature prominently in .NET Framework since it was introduced, and features have been added to the runtime just to support LINQ. Although this isn't a LINQ exam by any stretch, a good knowledge of LINQ and how it works is something I can all but promise you will be rewarding, both at work and on the exam.

The coverage of LINQ-to-XML is covered after the coverage of the primary System.Xml namespace classes. This is not an accident. Other than some tweaks and small improvements, the System.Xml namespace in version 4.0 or version 4.5 of the Framework is still very similar to what it was in earlier versions. There's not a lot of new material to cover there, so although it is certainly fair game for the exam, it's doubtful that you'll find a whole lot of emphasis on it. I can assure you, however, that LINQ-to-XML will be covered on the exam.

Coverage of System.Xml preceded LINQ-to-XML because the hope was to drive home how awkward XML parsing using traditional means is (and although the traditional means might be awkward or inelegant, they are much more elegant than the alternatives of the time) by juxtaposing it against the elegance and simplicity that LINQ-to-XML provides.

To take advantage of it, note that, to provide the features it does, it takes much advantage of the more modern aspects of each .NET language and the .NET Framework, such as each of these:

- Anonymous methods

- Generics
- Nullable types
- LINQ query semantics

To begin the discussion, let's start with where everything here lives. You'll find the classes for the LINQ-to-XML API in the System.Xml.Linq namespace.

The XElement class is one of the core classes of the LINQ-to-XML API and something you should be familiar with. It has five constructor overloads:

```
public XElement(XName someName);
public XElement(XElement someElement);
public XElement(XName someName, Object someValue);
public XElement(XName someName, params Object[] someValueset);
public XElement(XStreamingElement other);
```

Remember what you had to do before with the XDocument class to create a Customers element: a Customer element and then a FirstName, MiddleInitial, and LastName element corresponding to it. (To emphasize the difference, you might want to refer to the previous section if you can't immediately visualize it.)

Now let's look at the same process using just the XElement class:

```
XElement customers = new XElement("Customers", new XElement("Customer",
                new XElement("FirstName", "John"), new XElement("MiddleInitial", "Q"),
                new XElement("LastName", "Public")));
```

The code snippet produces the following output:

```
<Customers>
  <Customer>
     <FirstName>John</FirstName>
     <MiddleInitial>Q</MiddleInitial>
     <LastName>Public</LastName>
  </Customer>
</Customers>
```

That's just the beginning. You can easily reference items inside the node. Although these are all strings, you can easily cast them to different .NET types as needed if you query against a different object type or structure. Examine the following code:

```
XElement customers = new XElement("Customers", new XElement("Customer",
                new XElement("FirstName", "John"), new XElement("MiddleInitial", "Q"),
                new XElement("LastName", "Public")));
String fullName = customers.Element("Customer").Element("FirstName").ToString() +
                customers.Element("Customer").Element("MiddleInitial").ToString() +
                customers.Element("Customer").Element("LastName").ToString();
```

This code produces the corresponding output:

```
<FirstName>John</FirstName><MiddleInitial>Q</MiddleInitial><LastName>Public</LastName>
```

NOTE **XELEMENT AND ESCAPE CHARACTERS**

Like any language, XML has characters that define it. You might come across situations in which you need to use those characters (and many of them happen to be characters that are used quite often). Greater-than and less-than symbols are part of the language syntax; ampersands are as well. Normally, you have to manually escape these. &gt; is used instead of a greater-than symbol; &lt needs to be used in place of a less-than symbol; &amp replaces an ampersand character. XElement automatically calls the XmlConvert class, which escapes these for you. As such, the following code snippet would work just fine:

```
    XElement customers = new XElement("CustomerNote", "The customer
really likes to use < and  > in their correspondence. They love using &
and </div> as well");
```

The XDocument class almost seems redundant when compared with the XElement class. To quote MSDN:

> *The XDocument class contains the information necessary for a valid XML document. This includes an XML declaration, processing instructions and comments. Note that you only have to create XDocument objects if you require the specific functionality provided by the XDocument class. In many cases you can work with the XElement. Working directly with XElement is a simpler programming mode.*

The following list summarizes the basic components of an XDocument instance:

- **One XDeclaration object** The declaration enables you to specify the version of XML being used, the encoding, and whether the document contains a document type definition.
- **One XElement object** Because a valid document must contain one root node, there must be one XElement present. Note that, although you need to use an XElement to use an XDocument, the reverse is not the case.
- **XProcessingInstruction objects** Represents an XML processing instruction.
- **XComment objects** As with XProcessingInstruction, you can have one or more. According to MSDN, the only caveat is that this can't be the first argument in the constructor list. Valid documents can't start with a comment. The irony here is that there are no warnings generated if you use it as the first argument; the document parses correctly, and MSDN's own examples show example after example that specifically violate this rule. Answers to questions on the exam have to be 100 percent correct and provable. If you look around the Internet, there's a good bit of debate about this subject. It's doubtful you'll see questions on the exam that differentiates the correct versus incorrect answer. Suffice to say that, although MSDN says one thing, its own examples directly violate this rule, so if you did happen to fail the exam by this one question, you can certainly argue your case.

Because the issue is possibly confusing, the following example shows passing in an XComment as the first argument and the output:

Code:

```
XDocument sampleDoc = new XDocument(new XComment("This is a comment sample"),
    new XElement("Customers",
        new XElement("Customer",
            new XElement("FirstName", "John"))));
sampleDoc.Save("CommentFirst.xml");
```

Output:

```
<?xml version="1.0" encoding="utf-8"?>
<!--This is a comment sample-->
<Customers>
  <Customer>
    <FirstName>John</FirstName>
  </Customer>
</Customers>
```

The XAttribute class is so simple that it doesn't need much discussion. To declare one, you simply instantiate it, passing in the name of the attribute and the value:

```
XAttribute sampleAttribute = new XAttribute("FirstName", "John");
```

Attributes, by definition, have no meaning outside of the context of an element, so they are obviously used only in conjunction with an XElement.

XNamespace is easy to create and work with:

```
XNamespace billCo = "http://www.billco.com/Samples";
```

If you want to use an XNamespace in conjunction with an XElement, you simply append it to the Element Name. The following illustrates both the declaration and how to use it in conjunction with an XElement:

```
XNamespace billCo = "http://www.billco.com/Samples";
XElement firstNameBillCo = new XElement(billCo + "FirstName", "John");
```

This is a subject that has endless possibilities and permutations and is impossible to illustrate completely. The main thing to understand is how to create an XNamespace and how to use it.

Declaring and instantiating each of the X classes isn't complicated. However, things get tricky when it comes to queries. Let's look at the following code snippet:

```
String documentData = @"<Customers><Customer><FirstName>John</FirstName></Customer>
            <Customer><FirstName>Bill</FirstName></Customer>
            <Customer><FirstName>Joe</FirstName></Customer></Customers>";
XDocument docSample = XDocument.Parse(documentData);
var descendantsQuery = from desc in docSample.Root.Descendants("Customer")
                        select desc;
var elementsQuery = from elem in docSample.Root.Elements("Customer")
                    select elem;
Int32 descendantsCount = descendantsQuery.Count();
Int32 elementsCount = elementsQuery.Count();
```

```
Console.WriteLine(descendantsCount.ToString());
Console.WriteLine(elementsCount.ToString());
```

The output in both cases is 3. From a behavioral point of view, they look identical, don't they? Search on MSDN and see the way they are each defined and described in almost identical terms. The way to think of it is this: Descendants return whatever elements you choose from the entire subtree of each element. Elements, on the other hand, yield only the child elements. There are 1,000 different examples to illustrate this point, but they are more confusing than helpful. The only time it matters is when there are not child elements inside one of the ones you are searching for. It behaves differently only if there are child elements that also have the same name as the element you are looking for. The same queries run against this snippet yield the same results as they did originally, even though there are nested elements:

```
String documentData = @"<Root><CustomerName><FirstName>John</FirstName></CustomerName>
                                    <CustomerName><FirstName>Bill</FirstName>
</CustomerName>
                                    <CustomerName><Other><Another>Blah</Another>
</Other><FirstName>Joe</FirstName>
<MiddleInitial>Q</MiddleInitial>
                                              <LastName>Public</LastName>
</CustomerName></Root>";
```

Make a slight modification, in which the search element name is used in a nested element, and you have totally different behavior:

```
String documentData = @"<Root><CustomerName><FirstName>John</FirstName></CustomerName>
                        <CustomerName><FirstName>Bill</FirstName></CustomerName>
                                    <CustomerName><Other><CustomerName>Blah
</CustomerName></Other><FirstName>Joe</FirstName>
<MiddleInitial>Q</MiddleInitial>
                                              <LastName>Public</LastName>
</CustomerName></Root>";
```

There are countless extension methods, but most are self-explanatory or used rarely enough so they shouldn't present much of a problem from an exam perspective. Just remember the difference in behaviors between these two because questions are likely to appear on the exam.

Advanced XML manipulation

If you know only LINQ, and you are an absolute expert, you'll probably do well on this portion of the exam (but don't take that as any sort of endorsement to not study the System.Xml.Linq namespace objects). Recall that you have to specifically escape several reserved characters and sequences unless you are using a class, method, or property that does it for you. So the first thing to know is that XmlConvert automatically escapes reserved items. It also does more than that. Think of the Convert class in the System namespace. It has several methods, such as ToInt32, ToDateTime, ToBoolean, and many more. Think of the XmlConvert class as its Xml obsessed sibling. XmlConvert.ToDateTime, XmlConvert.ToDecimal, XmlConvert.ToGuid, and any other method that contains "To," followed by a framework type, should be self-explanatory, but some aren't as intuitive (see Figure 1-8). Again, take a look at the class, run through it, and familiarize yourself with as much as possible; then determine whether things are valid or invalid. Table 1-12 covers several of these methods (this list is not comprehensive, but covers some of the less intuitive methods you might encounter).

TABLE 1-12 XmlConvert class methods

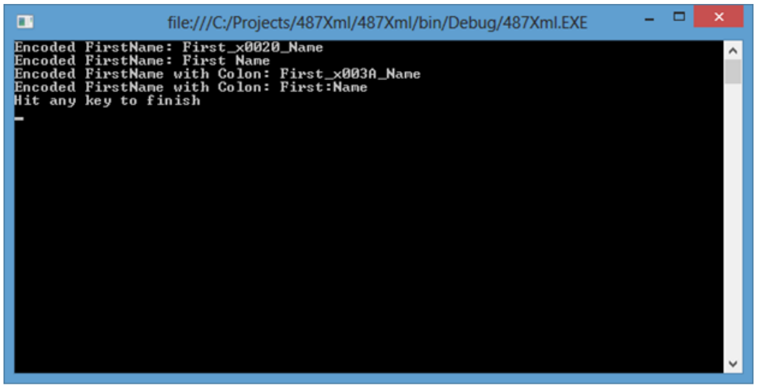
Name	Behavior
DecodeName	Decodes the name of an item that's been encoded already. The samples that follow this table illustrate it.
EncodeName	Encodes a string so that it can definitely be used as an XML node name.
EncodeLocalName	Behaves virtually identically to EncodeName, but with one major difference: It actually encodes the colon character, which ensures that Name can be used as the local name element of a namespace-qualified name. Again this method will be emphasized in the following code sample.
EncodeNmToken	Returns a valid name for a node according to the XML Language spec. It might sound identical to EncodeName, and they are very similar. The primary difference that EncodeNmToken encodes colons wherever they appear, which means that it could return a "valid" element according to syntax rules, but not according to the namespace specification.
IsStartNCNameChar	Determines whether the parameter is a valid non-colon-character type.
IsPublicIdChar	If the parameter is a valid public identifier, it returns it; otherwise, it returns null. If you're not familiar with XML public and system identifiers, try entering "XML Public Identifier" into Bing—there are plenty of links available on it. In reality, however, public identifiers are just magic strings and mainly exist because of legacy behavior more than any necessity.
ToDateTimeOffset	Represents a specific point in time with respect to *Coordinated Universal Time (UTC)*. There are three overloads for this method: one that just accepts a string, one that accepts a string and a second string that represents the format the date is represented in within the string, and the last that accepts a string and an array of strings containing the formats.

There are countless overloads for members, such as ToInt32 or ToDateTime, and they behave just as you expect and are easy to follow. The following code illustrates the encoding and decoding issues, which are the most relevant and what you're most likely to run into on the exam:

Code

```
String encodedFirstName = XmlConvert.EncodeName("First Name");
Console.WriteLine("Encoded FirstName: {0}", encodedFirstName);
String decodedFirstName = XmlConvert.DecodeName(encodedFirstName);
Console.WriteLine("Encoded FirstName: {0}", decodedFirstName);
String encodedFirstNameWithColon = XmlConvert.EncodeLocalName("First:Name");
Console.WriteLine("Encoded FirstName with Colon: {0}", encodedFirstNameWithColon);
decodedFirstName = XmlConvert.DecodeName(encodedFirstNameWithColon);
Console.WriteLine("Encoded FirstName with Colon: {0}", decodedFirstName);
```

The output is shown in Figure 1-8.



```
Encoded FirstName: First_x0020_Name
Encoded FirstName: First Name
Encoded FirstName with Colon: First_x003A_Name
Encoded FirstName with Colon: First:Name
Hit any key to finish
_
```

FIGURE 1-8 Encoding and decoding using the XmlConvert class

Thought experiment

Creating an XML manipulation strategy

In the following thought experiment, apply what you've learned about the "Manipulate XML data structures" objective to determine the data access strategy for new application development at your company. You can find answers to these questions in the "Answers" section at the end of this chapter.

You are building an application that makes extensive use of XML. The document structures follow very consistent patterns, and sometimes large applications to manipulate these structures are similar.

With this in mind, answer the following questions:

1. Would your application benefit from using LINQ-to-XML?
2. What could be done to simplify consumption of the documents?

Objective summary

- The XElement and XDocument classes are the primary or topmost objects you'll typically be working with, although XElement is frequently the better choice.
- Although very similar in most regards, the XDocument represents an entire XML document, whereas the XElement represents an XML fragment. The practical differences between those two are often trivial or nonexistent, but it's worth noting.
- Escaping reserved characters and dealing with namespaces are the two other nuances you'll probably encounter on the exam. You don't need to be an expert in either, but you should at least be familiar with the XmlConvert and XmlNamespace classes and what they are used for.

- Although there are different classes and methodologies in the .NET Framework regarding XML manipulation, don't think that you have to use either one approach (System.Xml) or the other (System.Xml.Linq). LINQ is built into the Framework, as its name indicates, and you can certainly use an XmlWriter class and then query a structure using LINQ-to-XML features. (The point is that you can mix and match as you see fit or as needed; using one methodology doesn't force your hand in dealing with the other.)
- Make sure that you understand the difference in behavior between the Elements and Descendants methods of the XContainer class.

Objective review

Answer the following questions to test your knowledge of the information in this objective. You can find the answers to these questions and explanations of why each answer choice is correct or incorrect in the "Answers" section at the end of this chapter.

1. You need to use an ampersand in the name of an element you create with an XElement item. Which of the following allows a correctly created element name?
   - **A.** Just pass in the string to the XElement constructor; it handles it automatically by calling the XmlConvert class transparently.
   - **B.** Manually escape the character sequence.
   - **C.** Use the XmlConvert.Encode method to create the string to pass into the constructor.
   - **D.** All of the above.
2. Which of the following are requirements for a valid and well-formed XML document?
   - **A.** Every element must have a corresponding closing element.
   - **B.** Every element must have at least one attribute.
   - **C.** Every attribute must have a corresponding closing attribute.
   - **D.** Elements and attributes can be used interchangeably as long as they have open and closing tags.

Chapter summary

- Microsoft provides developers with several different ways to access and manipulate data. These range from the older and more traditional to more cutting edge. ADO.NET, the EF, and ADO.NET Data Services are the three core areas that handle data access.
- Each of the data access methods provides many of the same features (with limited exceptions), but the ease with which they do it and tooling support varies greatly.

- Constantly querying the database for items that are relatively static just produces unnecessary overhead on the database, network resources, and server resources. By implementing a caching strategy, much of that overhead can be reduced.
- Caching, although very beneficial in many cases, must be carefully considered, and the costs and benefits must be weighed carefully. You can fix one type of problem (performance) while introducing very serious other types of problems (stale data) if you're not careful. This problem might be trivial or completely catastrophic, depending on the application's requirements.
- In complex distributed systems, transactions are more a necessity than a luxury. Transactions allow things to happen in an all-or-nothing fashion and enable you to walk back failures and mistakes without having to resort to costly and serious measures such as database restores.
- There are two types of transactions: simple and distributed. Simple transactions cover one connection to one data source; distributed transactions cover connections to one or more sources.
- You can implement transactions with the System.Transactions.TransactionScope class or the SqlTransaction class (or OracleTransaction and other implementations, as the case may be). TransactionScopes provide the benefit of being able to transparently handle both simple and distributed transactions by invisibly promoting them when needed without any developer involvement. Compared to how difficult performing distributed transactions were just a few years ago, the simplicity the TransactionScope provides is nothing short of amazing.
- Windows Azure provides several means for data storage when requirements dictate cloud-based storage. For structured data, you have TableStorage for nonrelational data and SQL databases for relational data. For large unstructured binary data, you have Blob storage. For less-complex scenarios, you have local storage that provides per-instance temporary storage.
- WCF Data Services are the latest incarnation of ADO.NET Data Services. Using the EF as a backdrop to manipulate entities in the form of .NET classes, Data Services enables you to build fully functioning data services that enable users to employ a full array of options in data manipulation. It has the elegance of allowing queries to be URL addressable and has a rich variety of formats that data can be transmitted with, including Atom and JSON.
- With the popularity of XML, even many tools and libraries meant to deal with XML parsing are still awkward and discomforting for many. There are existing well-established libraries for manipulating XML data, and there are many more-modern libraries, too. On the one hand, you have the System.Xml namespace items; on the other, you have System.Xml.Linq. The beauty is that they can almost always be used in conjunction with each other if you prefer, so you're never bound to just one or the other. Virtually everything you could ever need or want to do with XML can be done with these two libraries.

Answers

This section contains the solutions to the thought experiments and answers to the lesson review questions in this chapter.

Objective 1.1: Thought experiment

1. If you continue to build using ADO.NET, meeting application requirements will be increasingly difficult. It would provide consistency, but any functionality that needs to be exposed to the outside world will take more and more effort (arguably wasted effort to accomplish).
2. In and of itself, the EF doesn't get you closer to the goal, but it would enable easier interoperability with WCF Data Services, for instance. It would set the groundwork for moving to OData.
3. Moving legacy applications to WCF Data Services would not be trivial. They are already built and tested. However, moving them so they could be consumed externally will require changes, no matter how it is accomplished. WCF Data Services would allow for all the required features that don't currently exist.

Objective 1.1: Review

1. **Correct answers:** A, B, C, D
   - **A.** **Correct:** This is technically correct, but dropping the table means that all the data goes with it. That could work in some development environments, but in production it is a terrible idea. Because the environment wasn't mentioned, this is still a correct answer.
   - **B.** **Correct:** This would update the entity and the data source correctly.
   - **C.** **Correct:** This would update the entity and the data source correctly.
   - **D.** **Correct:** This would update the entity and the data source correctly.
2. **Correct answer:** C
   - **A.** **Incorrect:** LINQ-to-SQL will facilitate data access, but does nothing to expose the data to the web.
   - **B.** **Incorrect:** The EF will facilitate data access, but does nothing to expose the data to the web.
   - **C.** **Correct:** This would enables you to meet all the data requirements as well as the transmission requirements.
   - **D.** **Incorrect:** This technology no longer exists.

3. **Correct answer:** D
    - **A.** **Incorrect:** This file technically no longer exists, so wouldn't be appropriate.
    - **B.** **Incorrect:** This file technically no longer exists, so wouldn't be appropriate.
    - **C.** **Incorrect:** This file technically no longer exists, so wouldn't be appropriate.
    - **D.** **Correct:** Each of the preexisting files, including the mapping files, is included here.

Objective 1.2: Thought experiment

1. The more static and less volatile it is, the better the candidate is for caching.
2. How frequent repeat trips are to the source coupled with the size of it. At some point, reducing trips to the database can be offset by size of the data. Local storage or cache storage sizes can be burdensome, so caching isn't a magic bullet.
3. The more frequently data changes, the less likely it is to be a viable candidate for caching. Monitoring is not inexpensive, and although very helpful in some cases, it would offset the benefit of caching. In the same respect, if the expiration policy is very small, the benefit of caching is diminished. If monitoring needs to happen in most cases, any benefit from caching will be greatly reduced, and a scenario could arise in which things are much worse because of it. Such scenarios are atypical and usually arise from overuse, but they do happen.

Objective 1.2: Review

1. **Correct answers:** A, B, D
    - **A.** **Correct:** CacheEntryChangeMonitor is a valid option.
    - **B.** **Correct:** FileChangeMonitor is a valid option.
    - **C.** **Incorrect:** There is no such thing as an MsmqChangeMonitor.
    - **D.** **Correct:** SqlChangeMonitor is a valid option.
2. **Correct answer:** C
    - **A.** **Incorrect:** The System.Runtime.Caching.CacheItemPriority enum does not have a Normal option. It does exist, however, in the System.Web.Caching.CacheItem Priority enum.
    - **B.** **Incorrect:** The System.Runtime.Caching.CacheItemPriority enum does not have a High option. It does exist, however, in the System.Web.Caching. CacheItemPriority enum.
    - **C.** **Correct:** There is a NotRemovable option.
    - **D.** **Incorrect:** The System.Runtime.Caching.CacheItemPriority enum does not have a Low option. It does exist, however, in the System.Web.Caching. CacheItemPriority enum.

3. **Correct answers:** A, D
    - **A. Correct:** This would accomplish the requirements.
    - **B. Incorrect:** The second parameter cannot be a CacheItemPriority. It will not compile.
    - **C. Incorrect:** This does not set the Region name.
    - **D. Correct:** This would accomplish the requirements.

Objective 1.3: Thought experiment

1. Clearly defined boundaries are the main consideration. The more items affected in a transaction and the longer an operation takes, the more problems you'll see related to making it transactional.
2. There are several costs associated with implementing transactions, and these costs can add up quickly. If every operation were transactional, it would add tremendous cost without necessary benefit.
3. The importance of the data is one main consideration. How long-running the process is would be another consideration. The number of resources affected would be another one.

Objective 1.3: Review

1. **Correct answers:** B, D
    - **A. Incorrect:** ReadUncommitted would result in nonrepeatable reads.
    - **B. Correct:** RepeatableRead would be the loosest IsolationLevel that would meet these requirements.
    - **C. Incorrect:** Chaos does not work with SQL Server.
    - **D. Correct:** Snapshot would meet these requirements but might be overkill.
2. **Correct answer:** B
    - **A. Incorrect:** Long running operations that span several tables may necessitate transactional support, but the duration and complexity would generally pose serious locking and contention issues.
    - **B. Correct:** Small fast operations are generally the best candidates for transactions.
    - **C. Incorrect:** Operations that involve the file system would generally be problematic and would not benefit from transactions.
    - **D. Incorrect:** A non-Windows operating system would present many problems and would not be a good candidate for transactional support.

Objective 1.4: Thought experiment

1. Blog storage for the video and table storage for the statistical information would probably be best here.
2. It might be possible to use just one approach. However, although you could use Blob storage to store the statistical information, you couldn't use table storage for the media files in any practical sense.
3. If consumption is all clustered in well-defined areas, using CDN could be very beneficial.

Objective 1.4: Review

1. **Correct answer:** A
   - **A. Correct:** Chatroom data is mostly nonrelational conversations that have structure to them. This is a perfect candidate for Table storage.
   - **B. Incorrect:** Queues are not a good mechanism at all for this type of data because there is no way to query and fetch data based on those results.
   - **C. Incorrect:** Blob storage is a terrible option for this type of data because it would lose any structure the data had and would be on a slow medium.
   - **D. Incorrect:** The CDN would not work for this type of data.
2. **Correct answers:** B, C
   - **A. Incorrect:** CDN allows manipulation of nodes to reduce intermediate traffic. The whole idea is to have nodes closer to the traffic source. Little or no benefit will be realized if the traffic source is already close.
   - **B. Correct:** By specifying nodes closer to the traffic source, increased performance can be realized.
   - **C. Correct:** CDN helps distributed scale issues and is particularly well suited to traffic spikes and surges (such as ones associated with product launches).
   - **D. Incorrect:** CDN doesn't offer any security advantages that aren't already present in Azure offerings that do not take advantage of it.
3. **Correct answer:** A
   - **A. Correct:** An Azure CDN URL is in the format *http://<identifier>.vo.mscend.net/.*
   - **B. Incorrect:** This is a standard Azure storage URL.
   - **C. Incorrect:** This is a standard Azure storage URI with a misplaced identifier.
   - **D. Incorrect:** This is a completely malformed Azure storage URI.

Objective 1.5: Thought experiment

1. Yes, in fact, that's the primary benefit. Items can be accessed using traditional code or using URI-based semantics, but in this case, URI semantics would be particularly useful.
2. All Create, Retrieve, Update, and Delete (CRUD) operations.

Objective 1.5: Review

1. **Correct answer:** B
   - **A.** **Incorrect:** This would be right only if the data were being changed. It's just being queried.
   - **B.** **Correct:** This is the only one that can handle items when they are queried.
   - **C.** **Incorrect:** This simply would not work for the task at hand or even come close to it.
   - **D.** **Incorrect:** This simply would not work for the task at hand or even come close to it.
2. **Correct answer:** A
   - **A.** **Correct:** The entity name, followed by a filter condition using gt for greater than 5 is what's required here.
   - **B.** **Incorrect:** The query and filter syntax is incorrect.
   - **C.** **Incorrect:** No mention is made of a filter here, and the syntax is incorrect.
   - **D.** **Incorrect:** The id field and operator are correct, but the syntax is incorrect.
3. **Correct answer:** D
   - **A.** **Incorrect:** The filter and query syntax are incorrect.
   - **B.** **Incorrect:** The filter and query syntax are incorrect.
   - **C.** **Incorrect:** The filter syntax is incorrect.
   - **D.** **Correct:** The OrderBy field is specified correctly, as is the filter field.

Objective 1.6: Thought experiment

1. The more consistent the structures are, the more any approach will be simplified. LINQ semantics allow much simpler interaction with the documents, although any of the existing .NET XML libraries can work.
2. The more consistent the data, the more one approach can be reused. Because the documents are all similar, the hard part, so to speak, is largely addressed. You might consider doing an intermediate transform on the documents, though, to render them as one very consistent format that might make it even simpler.

Objective 1.6: Review

1. **Correct answer:** D
   - **A. Incorrect:** Technically correct, but other answers are all correct, so choice D is the only one.
   - **B. Incorrect:** Technically correct, but other answers are all correct, so choice D is the only one.
   - **C. Incorrect:** Technically correct, but other answers are all correct, so choice D is the only one.
   - **D. Correct:** Because each of the other answers is correct, this one is the right answer.
2. **Correct answer:** A
   - **A. Correct:** An element can self-close, but each opening element must be closed.
   - **B. Incorrect:** Elements might have attributes, but there is no requirement for an element to have an attribute.
   - **C. Incorrect:** Attributes don't require matching closure tags.
   - **D. Incorrect:** Elements can be nested and structured in a way that accomplishes the same goal as attributes, but attributes can't contain elements.

Index

About the authors

BILL RYAN is a Software Architect at Dynamics Four, a Microsoft Gold Partner and one of the nation's most innovative Dynamics CRM consultancies. He lives in Greenville, SC, with his wife and daughter. Bill has won Microsoft's Most Valuable Professional award 9 years in a row in several different product groups and was one of 20 people invited into Microsoft's Data Access Insiders program. Bill has been actively involved in Microsoft's exam development for the last seven years, serving as a subject matter expert in various technologies including SQL Server Administration, Business Intelligence, .NET, Windows Communication Foundation, Workflow Foundation, and Windows Phone. Outside of technology, he spends his time listening to the Phil Hendrie show and wrangling the many rescue pups he and his wife have fostered.

WOUTER DE KORT is an independent technical coach, trainer, and developer at Seize IT. He is MCSD certified. As a software architect, he has directed the development of complex web applications. He has also worked as a technical evangelist, helping organizations stay on the cutting edge of web development. Wouter has worked with C# and .NET since their inception; his expertise also includes Visual Studio, Team Foundation Server, Entity Framework, Unit Testing, design patterns, ASP.NET, and JavaScript.

SHANE MILTON is a Senior Architect creating enterprise systems running in Windows Azure and is currently designing cloud-based Smart Grid solutions to manage energy for millions of homes and businesses throughout the US. As an active leader in training and educating teams in cloud technologies and various Agile techniques, he takes particular interest in offering his expertise to community user groups and regional conferences in and around Indianapolis.

Now that you've read the book...

Tell us what you think!

Was it useful?
Did it teach you what you wanted to learn?
Was there room for improvement?

Let us know at http://aka.ms/tellpress

Your feedback goes directly to the staff at Microsoft Press, and we read every one of your responses. Thanks in advance!

Microsoft