Chapter 1

Getting to Know LINQ

In This Chapter

- Defining LINQ uses, benefits, and design goals
- Considering the real world uses of LINQ
- Defining declarative programming languages
- Understanding the LINQ namespaces

The Language INtegrated Query (LINQ) feature of Visual Studio 2008 provides you with a new way to interact with data of all types. In fact, this new feature provides you with tools that make it easier to create queries using less code. The resulting queries are often easier to understand than other techniques for deriving information from both standard (think databases) nonstandard (think memory data structures) data sources. In addition, you gain a measure of flexibility that most developers associate with using a database, not lists provided internally as part of applications.

The easiest way to think of LINQ at the outset is as a means of looking for something — a specialized kind of search. Because most people are inundated with information today, providing a fast means of locating specific data is important. LINQ provides the means to perform a search without writing a lot of code. Everything is built in to the development environment so all you need to consider is what to find, not how to find it. Unlike other kinds of searches, however, LINQ provides the means to look inside data structures that you normally can't search, such as objects. It can also standardize the methods you use to perform searches within Web services. In short, LINQ

- Provides access to a huge range of data
- Lets you simplify searches to locate just what you need
- Reduces the code required to perform a search
- Enables you to focus on the search instead of writing search routines
- Interacts with all kinds of data sources using a standardized approach

This chapter serves as an introduction to LINQ. You discover how LINQ will make your coding experience better, reduce real world complexity, and make searches more accurate. As part of discovering LINQ, you also need to know about declarative languages, and this chapter provides the information you need. Finally, since LINQ is part of the .NET Framework, you need to know which namespaces support it, so this chapter provides an introduction to these new namespaces.

Considering LINQ

LINQ is possibly the most exciting new feature Microsoft has added to Visual Studio 2008. Sure, the other features that Microsoft added are important, but they don't have the overwhelming reach of LINQ to change the way developers write applications. Anyone can use LINQ to create a better application — one that works more efficiently and uses less code. In addition, you no longer have to write custom search routines that differ from developer to developer. By using LINQ to perform searches of all types, you can standardize another part of your code base and incrementally improve overall developer productivity. The following sections describe LINQ in greater detail.

Understanding the task that LINQ performs

LINQ is all about searching efficiently and consistently. Your application searches efficiently by performing the task using less code and obtaining the results faster. Consistency comes from using the same code pattern to perform a search no matter what source of data you want to work with. From a pattern perspective, a search of an array looks the same as a search of a Web service or SQL Server database. Using LINQ, it no longer matters whether the data resides in SQL Server or MySQL, or even both. LINQ does divide queries into four common types (using different providers) that augment the basic patterns described in Chapter 2:

- LINQ to Object
- LINQ to DataSet
- LINQ to SQL
- LINQ to XML

It's possible to have other kinds of "LINQ to" scenarios by adding other libraries. For example, you can find a LINQ to Active Directory library at `http://www.codeplex.com/LINQtoAD`. The goal, however, is to perform

as many tasks as possible using the four basic LINQ to strategies provided with .NET Framework 3.5.

The most important thing to remember about LINQ is that it isn't technology specific. This book shows you how to "LINQ to" any number of data sources, some of which you'll find unusual because you may not have thought to search them before. For example, you may have an assortment of data in an object that you need to search — LINQ is the perfect tool for performing this task.

In addition to finding data, LINQ can also help organize it so that you present the user with only the data needed as output. Using special features of LINQ, you can filter data so that the user sees just the desired elements. You can also group and sort the data so that the user sees it in an order that makes it easier to use the data. The essential task that LINQ performs, therefore, is to make the data accessible. The user sees only the data needed and in the most productive way.

However, LINQ goes beyond searching for and ordering data. In many cases, you can also use LINQ to create a query that manipulates data in various ways, assuming the data source allows such manipulation. For example, you can use LINQ to change the content of a SQL Server database. Some unusual data sources such as Active Directory also allow modification. In fact, you can modify any configuration database that relies on the Lightweight Directory Access Protocol (LDAP). Consequently, the techniques in this book show you how to work with mainstream products, but you can easily modify them to meet any need.

Contemplating why you need LINQ

With feature bloat running rampant and developer time in ever limited quantity, you may wonder whether LINQ is the right choice for you. In most cases, you'd look for a list of qualifiers describing the technology and use these qualifiers to decide whether a technology is the right one for you. LINQ is a well-designed technology that can apply to anyone's search needs — no qualifiers needed. You won't have to wait for drivers or additional software to use it. In short, anyone who searches for data can use LINQ to meet that need.

Of course, now you're thinking that this book is offering you the fabled silver bullet solution. LINQ isn't a silver bullet. The other tools you have for searching are still useful and you'll need to employ them. For example, even Microsoft admits that LINQ can have performance problems when searching SQL Server databases. You can read the five-part blog series about performance issues at `http://blogs.msdn.com/ricom/archive/2007/06/22/dlinq-linq-to-sql-performance-part-1.aspx` to obtain a good

overview of the problems (but not any significant solutions). Chapter 11 provides you with information about how you can overcome performance issues (here's where you find the solutions), so make sure to check it out as well.

It's important to understand that LINQ works with most data sources but not all of them. For example, you can easily use LINQ with most public Web services and some private Web services. However, even though you can use LINQ with the Amazon, Google, and AOL Web services, it doesn't work with the eBay Web services due to security concerns. In short, specific Web service requirements can prevent LINQ from working properly. You can find a complete list of LINQ providers (LINQ to solutions) at `http://blogs.msdn.com/charlie/archive/2006/10/05/Links-to-LINQ.aspx`. Table 1-1 shows a list of the providers as of this writing.

Table 1-1 LINQ Providers

LINQ to Solution	*URL*
LINQ Extender (toolkit for building LINQ providers)	`http://www.codeplex.com/LinqExtender`
LINQ over C# project	`http://www.codeplex.com/LinqOverCSharp`
LINQ to Active Directory	`http://www.codeplex.com/LINQtoAD`
LINQ to Amazon	`http://weblogs.asp.net/fmarguerie/archive/2006/06/26/Introducing-Linq-to-Amazon.aspx`
LINQ to Bindable Sources (SyncLINQ)	`http://paulstovell.net/blog/index.php/why-synclinq-should-matter-to-you/`
LINQ to CRM (Customer Relationship Management)	`http://www.codeplex.com/LinqtoCRM`
LINQ to Excel	`http://www.codeplex.com/xlslinq`
LINQ to Expressions (MetaLinq)	`http://www.codeplex.com/metalinq`
LINQ to Flickr	`http://www.codeplex.com/LINQFlickr`
LINQ to Geo (geospatial data)	`http://www.codeplex.com/LinqToGeo`
LINQ to Google	`http://www.codeplex.com/glinq`
LINQ to Indexes	`http://www.codeplex.com/i4o/Release/ProjectReleases.aspx?ReleaseId=3519`

LINQ to Solution	*URL*
LINQ to IQueryable	`http://blogs.msdn.com/mattwar/archive/2007/08/09/linq-building-an-iqueryable-provider-part-vi.aspx`
LINQ to JavaScript	`http://www.codeplex.com/JSLINQ`
LINQ to JSON (JavaScript Object Notation)	`http://james.newtonking.com/archive/2008/02/11/linq-to-json-beta.aspx`
LINQ to LDAP (Lightweight Directory Access Protocol)	`http://community.bartdesmet.net/blogs/bart/archive/2007/04/05/the-iqueryable-tales-linq-to-ldap-part-0.aspx`
LINQ to Lucene	`http://www.codeplex.com/linqtolucene`
LINQ to Metaweb (freebase)	`http://www.codeplex.com/metawebToLinQ`
LINQ to MySQL, Oracle, and PostgreSql	`http://code2code.net/DB_Linq/`
LINQ to NHibernate	`http://www.ayende.com/Blog/archive/2007/03/17/Implementing-Linq-for-NHibernate-A-How-To-Guide--Part.aspx`
LINQ to RDF (Resource Description Framework) Files	`http://blogs.msdn.com/hartmutm/archive/2006/07/24/677200.aspx`
LINQ to SharePoint	`http://www.codeplex.com/LINQtoSharePoint`
LINQ to SimpleDB	`http://www.codeplex.com/LinqToSimpleDB`
LINQ to Streams	`http://www.codeplex.com/Slinq/`
LINQ to WebQueries	`http://blogs.msdn.com/hartmutm/archive/2006/06/12/628382.aspx`
LINQ to WMI (Windows Management Instrumentation)	`http://bloggingabout.net/blogs/emile/archive/2005/12/12/10514.aspx`, `http://tomasp.net/blog/linq-expand.aspx`, and `http://tomasp.net/blog/linq-expand-update.aspx`
LINQ to XtraGrid	`http://cs.rthand.com/blogs/blog_with_righthand/archive/2008/02/23/LINQ-to-XtraGrid.aspx`

Don't get the idea that LINQ always requires a provider. The provider does make it easier to perform tasks, but you can also create your own interface using the generic providers. For example, you can interact with Office 2007 files without using a specific provider. Chapter 10 shows you how to perform this task.

Even with these few warts, however, LINQ is a good solution for many search needs and you should at least try it. You need LINQ because it has so much to offer and doesn't require a lot of time to master or use. LINQ provides the experimental platform that most developers crave. In those few situations where LINQ can't do a good job for you, experimentation can at least help you understand the data source better.

Defining the LINQ design goals

Microsoft had a number of design goals in mind when it created LINQ. These design goals affect how you view LINQ today and how you can use it to solve specific application development problems. The following list describes the design goals.

- **Data source access simplification:** One of the major issues of working with any data source is that the developer must know several disciplines to perform the task of accessing the data. For example, when working with SQL Server, the developer must understand the nuances of the base programming language, a database provider, and a language such as SQL to obtain access to the data. If the developer decides to access XML data, it's necessary to master an entirely different set of disciplines. LINQ overcomes this problem by providing a single method of accessing data.
- **Data manipulation simplification:** When you make a query using C# or Visual Basic .NET, you have to worry about the structure of the data source. For example, when you query SQL Server, you must consider the tables, indexes, views, and other structural elements of the database. The use of these structural elements is necessary but not helpful. You end up thinking about the data structure and not the data. Consequently, many developers create convoluted and difficult to understand data-manipulation code when what they really wanted was the data (the underlying structure isn't important).
- **Data translation:** In most cases, you must write special routines to move data from one data source to another. For example, if you want to move data from an XML file to SQL Server, you must perform some special tasks to do it. In addition, moving the data doesn't always provide the results you expected. Differences in data source capabilities make the translation less than perfect. LINQ reduces the complexity of data translation significantly. It doesn't always provide a perfect translation either, but you'll find that the translation is usually better because each provider performs the required translation for you.

- **Object mapping:** Most programming languages today rely on some form of object orientation. Objects have special characteristics that you won't find in many data sources. For example, an object doesn't respect the tables, indexes, and other data structures found in SQL Server. Consequently, you need a means of mapping the object to the data source and vice versa. In the past, the developer had to rely on complex objects that Visual Studio generated for them. Using LINQ provides object mapping without the complexity.
- **Language extensibility:** Microsoft provides a limited number of "LINQ to" providers as part of the .NET Framework. These providers are capable, but they don't address every need. Consequently, one of the design goals for LINQ is to provide language extensibility so that third parties can create other providers. Table 1-1 shows an example of just how many providers have already been created by third parties, and you can expect more in the future.
- **Multiple data source extensibility:** Because one of the goals for LINQ is data translation, it's important to have providers that can work with multiple data sources. The goal is to make it possible to move data from any data source to any other data source. In addition, Microsoft wants LINQ to be able to use data from any data source and combine it with data from any other data source to create a composite output.
- **Type safety:** A major problem with many data source usage scenarios today is that data problems are discovered only at run time, often without any help from the language product or the application. A developer may not discover a problem until someone complains about mangled data. The type safety features of LINQ help you discover potential data problems during compile time, when they're easy to fix, rather than getting your bad news later.
- **IntelliSense support:** Creating a query using standard development tools can be hit or miss. LINQ provides IntelliSense support so that you can see how to create the query as you create it.
- **Debugger support:** Due to strong typing and other features of LINQ, you get full debugger support, which makes finding a particular problem considerably easier. No longer do you have to look for that errant bit of code in a loop or the missed type issue in a custom class. LINQ helps you diagnose problems quickly and easily.
- **Older product support:** Even though most of this book uses new technology that Microsoft provides, you can use LINQ with older products as well. Obviously, the support isn't built in to these older products, so LINQ doesn't work as seamlessly. (You must use at least the .NET Framework 2.0.)
- **Backward compatibility:** One of Microsoft's major goals was to ensure that you could continue using all the data structures you used in the past. LINQ simply provides a different way to interact with those data structures.

Using LINQ with other languages

Don't get the idea that LINQ is going to remain a solution for C# and Visual Basic .NET developers alone. Using LINQ does require the use of a different compiler, but that won't stop other languages from employing it. You can already find support for LINQ in Microsoft's new F# language and you'll probably find it in use with C++ as well.

LINQ will appear as part of other language packages in the future. There are rumors that Borland Delphi will also have LINQ support at some time (read more at `http://www.eweek.com/c/a/Application-Development/Borland-Plans-to-Support-MS-LINQ-in-Delphi-Platform/`), and you can expect that other languages such as PHP will have it as well. If you want to find out more about languages that will support LINQ, check out Charlie Calvert's Community Blog at `http://blogs.msdn.com/charlie/archive/2006/10/05/Links-to-LINQ.aspx`.

Understanding the LINQ requirements

LINQ is part of .NET Framework 3.5. Consequently, you need Visual Studio 2008 to work with LINQ effectively. This book assumes that you have a copy of Visual Studio 2008 installed on your system. The examples rely on Visual Studio 2008 Professional Edition and you may not get precisely the same results when you use a different edition of the product. Theoretically, you could use LINQ with Visual Studio 2005 (Chapter 5 discusses this technique), but the bulk of this book relies on Visual Studio 2008.

Microsoft has also decided to focus attention on C# as the programming language of choice when using LINQ. C# provides a few extensions and features that make working with LINQ easier. However, you can use Visual Basic .NET quite well with LINQ, too. Although most of the techniques in this book work with any language you want to use, the example code appears in C#. The exception is Chapter 4, which shows how to work with LINQ with Visual Basic .NET. These examples will help you to apply any of the examples to the Visual Basic .NET environment.

The use of C# begs the question of what makes it so special. Chapter 3 describes the .NET Framework extensions that make working with LINQ considerably easier. Some of these language extensions are found only in C# and others are easier to work with in C#. You find a complete description of the features and how to use them in the chapter. For now, all you need to know is that this book will help you use LINQ no matter which language you choose and what platform you have. It's also important to know that the majority of the book is focused on Visual Studio 2008 C# developers because this is the group that Microsoft has chosen as its target group for LINQ.

Using LINQ in the Real World

Realistically, using LINQ is possibly overkill if your goal is to search through a short list of items found in a control. Most developers will use LINQ for something a little more complex than simple lists (then again, nothing stops you from using LINQ even for simple tasks — it's that fast and easy). You know from previous sections of the chapter that LINQ isn't a silver bullet solution. The technology has problems with security and you may not always find the performance stellar, so it's important to weigh the cost of using LINQ against the benefits it provides, which are substantial.

Many developers will likely begin using LINQ in places where they don't currently have a good solution, such as with Web services, or in situations where they already know how to perform a search, such as with SQL Server. The starting goal is to discover how well LINQ works to perform a basic query and then move on to something more complicated. Developers will want to kick the tires for a while and then discover that LINQ really does do powerful things with only a little code.

LINQ query testing is required

LINQ is a new technology. As such, it's tempting to look at the benefits and say that it's the new perfect tool or to look at the deficiencies and proclaim another Microsoft failure. However, after you begin working with LINQ, you begin to understand that LINQ is neither of these viewpoints — it's simply a new tool to put in your arsenal. Many developers will find that LINQ is one very good answer to specific needs, but as with any tool, it has limitations.

The problem now is that because LINQ is a new tool, you don't know anything about its limitations. This book presents a considerable number of examples, and it's a good idea to try them all. However, at some point, you're going to have to test LINQ against the tools you currently use or should use. For example, you should probably test LINQ against your current .NET code and the SQL Server stored procedures that you use. In some cases, LINQ is most definitely a winner, but in other cases, you'll want to stick with existing technologies.

A rule of thumb for LINQ is that it simplifies queries. If your goal is to simplify the task of querying a data source, LINQ is normally going to come out ahead. Because LINQ uses a standardized method to create a query, it's simple to learn, and that can also make it considerably much more reliable than existing technologies. Developers are less likely to make errors when they have a tool that makes writing code easy. The fact that LINQ queries are generally shorter than any code you can write also tends to reduce errors and make code more reliable. However, because you're depending on LINQ to determine how to perform a particular task, LINQ doesn't always provide the required performance, which is why you must test any solution you create against the existing model (when one exists).

The most exciting use of LINQ is to perform data translation. Currently, a query of Amazon's Web service, interpretation, and translation into a form that SQL Server will use can require several hundred lines of code. I know this from experience because I've written such code in the past. It's nice to make two queries with LINQ that require perhaps twenty lines of code to perform the same task. As development environments become more complex and the number of data sources increase, developers will need the special talents of LINQ to perform their data translation for them.

It doesn't take long to realize that LINQ in the real world is all about getting the job done fast, reliably, and with fewer lines of code. In addition, the simplification that LINQ provides makes it possible to use data sources even when you aren't completely familiar with them. For example, a developer who normally works with SQL Server would need training to work with MySQL because the two products have differences. Because LINQ hides the differences between these two products, a SQL Server developer could possibly work with MySQL with little, if any, training. All that the developer would need to do is to ensure that the query is formed correctly — and IntelliSense even helps with that issue. In short, the real world view of LINQ is that it makes developers incredibly productive.

Understanding Declarative Programming

Most Visual Studio developers already understand imperative languages because C# and Visual Basic .NET are imperative languages. An *imperative* language describes how to solve a particular problem. You use an imperative language to write a procedure to answer a specific need. A user clicks a button and the button click event handler provides a procedure to respond.

Imperative languages assume that you know how to solve a problem, and in most cases, you do. However, sometimes you don't know how to solve a problem or the language itself has gaps that make a solution difficult. Obtaining data from a data source is one of those problems. In this case, you need a declarative language such as LINQ. When using a *declarative* language, you state the problem and let the language decide how to solve the problem. Other kinds of declarative languages include the Structured Query Language (SQL) used in SQL Server. In fact, you'll find that LINQ has similarities to SQL, even though the two languages aren't directly compatible.

Declarative languages can be divided into several groups, including logic, functional, and query languages. LINQ is in the query language group. Microsoft's new F# language is in the functional group (see my article at `http://www.devsource.com/cp/bio/John-Paul-Mueller/`). No matter which group a declarative language is in, the basic assumption is the same: A developer provides a problem and the language provides the method for solving that problem. In short, a declarative language defines a relationship between a problem and its solution.

Part of the strength of LINQ is that you can combine it with an imperative language such as C# or Visual Basic .NET to create a stronger whole. Using LINQ lets you rely on the language itself to solve certain problems, such as how to obtain the data you specify from a particular data source.

Despite Microsoft's declarations to the opposite, LINQ, like SQL, isn't a pure declarative language. For example, you can include functions as part of a LINQ query, so the language doesn't necessarily define a pure relationship between a problem and its solution — you can tweak the solution using the function. In addition, the order in which you define the problem affects the solution. These deviations from a pure declarative language are necessary to ensure that you receive the proper outcome of a query. For the purposes of this book, LINQ is a declarative language in the query language group that interacts with the C# or Visual Basic .NET imperative languages.

An Overview of the LINQ Namespaces

Microsoft chose not to provide a single LINQ namespace. The .NET Framework has a number of LINQ namespaces, each of which creates a different kind of data connection. The following sections describe the higher level LINQ namespaces in .NET Framework 3.5. You could find other LINQ libraries on the Internet for use with other connection types. Chapter 12 discusses one such library that you can use for accessing Active Directory.

System.Linq namespace

The `System.Linq` namespace contains all basic classes and interfaces that you use to work with LINQ. Every LINQ to solution relies on this namespace for basic support. As you'll see in later chapters, this is the one namespace that you always include when you want to use LINQ. The samples in Chapter 2 show initial usage of this namespace and you'll also see it in Chapters 3 through 5. Chapter 6 begins the full examination of this namespace as part of working with LINQ to objects. You can find out more about this namespace at `http://msdn2.microsoft.com/en-us/library/system.linq.aspx`.

System.Linq.Expressions namespace

The `System.Linq.Expressions` namespace contains the classes, interfaces, and enumerations used to create expressions. An expression is essentially a tree of nodes that define how a query works. For example, you can create a binary expression that defines how to subtract one number from another. A constant expression can define a constant value, and a named

parameter expression can define a value that receives data of a particular type. The essential expression types are

- `BinaryExpression`
- `ConditionalExpression`
- `ConstantExpression`
- `InvocationExpression`
- `LambdaExpression`
- `ListInitExpression`
- `MemberExpression`
- `MemberInitExpression`
- `MethodCallExpression`
- `NewArrayExpression`
- `NewExpression`
- `ParameterExpression`
- `TypeBinaryExpression`
- `UnaryExpression`

Chapter 3 begins the discussion of several expression types, but you'll find expressions used throughout this book. You can find out more about this namespace at `http://msdn2.microsoft.com/en-us/library/system.linq.expressions.aspx`.

System.Data.Linq namespace

The `System.Data.Linq` namespace contains the classes, structures, interfaces, and enumerations used for SQL database interactions. This is the basic namespace used for LINQ to SQL scenarios. It's important to remember that this is LINQ to SQL and not LINQ to SQL Server. The classes in this namespace help you perform a number of data manipulation tasks, including:

- SELECT data from the database
- UPDATE data found in the database
- DELETE records as needed in the database
- Interact with binary data
- Use and implement referential integrity rules
- Work with a particular table or other database objects (such as indexes)
- Translate the data from one data source to another

You'll find that you use this namespace for a number of LINQ to SQL scenarios. Chapter 8 begins the discussion of working with LINQ to SQL Server. Chapter 11 discusses a number of advanced LINQ to SQL Server topics.

This namespace comes into play also when you work with `DataSet` objects. A `DataSet` needs to work exclusively with an external data source; you can also use it to work with internal data sources, so this is an extremely flexible namespace. Discussions in Chapter 7 show how you can use LINQ to DataSet to work with `DataSet` objects in your application. Look at Chapter 13 if you want to see how this namespace can affect other LINQ to scenarios, such as LINQ to MySQL. You can find out more about this namespace at `http://msdn2.microsoft.com/en-us/library/system.data.linq.aspx`.

System.Data.Linq.Mapping namespace

The `System.Data.Linq.Mapping` namespace contains the classes and enumerations to map data between an imperative language such as C# or Visual Basic .NET and a declarative language such as SQL. It also comes into play when working with technologies such as XML. In short, you'll use this class when working with any external data source that has a different representation from the standard object-oriented view of data found in the .NET Framework. Coverage of this namespace begins in Chapter 7, but you'll find it used throughout the book. You can find out more about this namespace at `http://msdn2.microsoft.com/en-us/library/system.data.linq.mapping.aspx`.

System.Data.SqlClient namespace

The `System.Data.SqlClient` namespace contains the classes used to create a basic connection with SQL Server. Although you might use this namespace in a number of scenarios, you'll generally use it exclusively with SQL Server. The classes in this namespace help you perform the following tasks:

- Interact with SQL Server 2000
- Interact with SQL Server 2005
- Perform string pattern matching
- Perform data manipulation, especially with dates
- Create a basic SQL Server connection

The coverage of this namespace begins in Chapter 8, but you'll also find advanced features described in Chapter 11. You can find out more about this namespace at `http://msdn2.microsoft.com/en-us/library/system.data.linq.sqlclient.aspx`.

System.Data.SqlClient.Implementation namespace

The `System.Data.SqlClient.Implementation` namespace contains the class used to implement the SQL Server client functionality found in the `System.Data.SqlClient` namespace. You generally won't use the classes found in this namespace directly. The coverage of this namespace begins in Chapter 8, but you'll also find advanced features described in Chapter 11. You can find out more about this namespace at `http://msdn2.microsoft.com/en-us/library/system.data.linq.sqlclient.implementation.aspx`.

System.Xml.Linq namespace

The `System.Xml.Linq` namespace contains classes and enumerations used to interact with XML data of all type. When you think about the number of ways in which modern computer systems use XML data, this namespace covers a significant amount of ground. As with SQL Server, you can use the classes of this namespace to interact with XML files in a number of ways. The following list provides an overview of the kinds of interaction you can perform:

- Load XML from files or streams (basic input functionality)
- Serialize XML to files or streams (basic output functionality)
- Create XML trees from scratch using functional construction (data translation)
- Query XML trees using LINQ queries (basic data viewing and manipulation)
- Manipulate in-memory XML trees (advanced data manipulation)
- Validate XML trees using XSD (data verification)
- Combine all of these features to perform advanced data translation tasks and even move data to other data sources

Chapter 9 begins the discussion of this particular namespace. However, you'll also see it in other parts of the book, such as when working with Office 2007 files in Chapter 10. You can find out more about this namespace at `http://msdn2.microsoft.com/en-us/library/system.xml.linq.aspx`.