LINQ to XML lets you apply the LINQ query style that we saw in Chapter 27 to XML data. This is done through a set of classes that represent XML data in a way that is designed to accommodate LINQ to Objects queries. These classes are in the System.Xml.Linq namespace, and they make it easy to work with XML. Once you understand how these classes work and relate to one another, you can use LINQ queries to create powerful and simple XML handlers.

The LINQ to XML classes read all the XML data into memory. This is similar to the Document Object Model (DOM) approach, which you may have used in other programming languages. C# does provide support for DOM, but it has been eclipsed by the more flexible and expressive LINQ to XML features.

The first part of this chapter shows you how to create and work with XML documents using the classes in the System.Xml.Linq namespace. The second part of this chapter shows you how to use LINQ queries to create and manipulate objects using the classes covered in the first part. We've already done the hard work understanding LINQ in Chapter 27. In this chapter, the heavy lifting is all about the System.Xml.Linq classes. Table 29-1 provides the summary for this chapter.

Table 29-1. Quick Problem/Solution Reference for Chapter 29

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
<th>Listings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create an object to represent an XML element.</td>
<td>Use the XElement class.</td>
<td>29-1</td>
</tr>
<tr>
<td>Create an XML fragment.</td>
<td>Use the XElement class, and pass other XElement objects as constructor parameters.</td>
<td>29-2 through 4, 29-7</td>
</tr>
<tr>
<td>Create different types of XML node.</td>
<td>Use other classes in the System.Linq.Xml namespace, such as XAttribute and XComment.</td>
<td>29-5</td>
</tr>
<tr>
<td>Create a valid XML document.</td>
<td>Use the XDocument class.</td>
<td>29-6</td>
</tr>
<tr>
<td>Read and write XML files.</td>
<td>Use the XElement.Load, XElement.Save, XDocument.Load, or XDocument.Save methods.</td>
<td>29-8, 29-9</td>
</tr>
<tr>
<td>Process an XElement.</td>
<td>Use the Attributes, Descendants, or Elements methods or the Name and Value properties.</td>
<td>29-10 through 29-12</td>
</tr>
<tr>
<td>Modify an XElement.</td>
<td>Use XElement class modification methods described in Table 29-5.</td>
<td>29-13, 29-14</td>
</tr>
</tbody>
</table>
Problem | Solution | Listings
--- | --- | ---
Query an XML fragment. | Use the Elements or Descendants methods to get an IEnumerable<XElement> suitable for use as a query data source. | 29-15
Create XML from a query. | Project new XElement objects. | 29-16, 29-17
Create objects from XML. | Project new objects whose fields and properties are set using XElement values. | 29-18
Transform XML data. | Use the standard LINQ query operators with IEnumerable<XElement> data sources. | 29-19 through 29-25
Use a query to modify the XML objects in the query data source. | Use Parallel LINQ and the ForAll extension method. | 29-26, 29-27

### Using the LINQ XML Classes

The System.Xml.Linq namespace contains a set of classes that you can use to create or modify XML. LINQ to XML relies heavily on these classes, and we need to understand how they function before we can move on to XML-related LINQ queries.

### Creating XML Declaratively

At the heart of LINQ to XML is the XElement class from the System.Xml.Linq namespace. We will use this class, or one of its close relations, in most of our LINQ to XML operations. We can use this class directly to create XML, as Listing 29-1 demonstrates; this is known as declarative or functional XML creation.

**Listing 29-1. Using the XElement Directly**

```csharp
using System;
using System.Xml.Linq;

class Listing_01 {
    static void Main(string[] args) {
        // create XElements
        XElement myNameElement = new XElement("Name", "Orange");
        XElement mySizeElement = new XElement("Size", "Large");

        // print out the XElement objects
        Console.WriteLine(myNameElement);
        Console.WriteLine(mySizeElement); 
    }
}
```