

## Kapitel 10

# LINQ to XML

### **In diesem Kapitel:**

Einführung in LINQ to XML	196
Vergleich Visual Basic 8 und Visual Basic 9	196
XML-Literale – XML direkt im Code ablegen	197
Erstellen von XML mithilfe von LINQ	198
Abfragen von XML-Dokumenten mit LINQ to XML	200
IntelliSense-Unterstützung für LINQ to XML-Abfragen	201

## Einführung in LINQ to XML

XML steht für *Extensible Markup Language*, was auf Deutsch soviel wie *erweiterbare Auszeichnungssprache* bedeutet. XML ist eine Untermenge von SGML, welches bereits 1986 vom World Wide Web Consortium (W3C) standardisiert wurde.

Bevor wir uns mit dem eigentlichen Kapitelthema *LINQ to XML* beschäftigen, lassen Sie uns kurz XML als solches rekapitulieren: XML besteht, vereinfacht gesagt, aus Elementen, Attributen und Werten.

**BEGLEITDATEIEN** Unter `.\Samples\Chapter10 - LinqToXml\LinqToXmlDemo` finden Sie die Beispieldateien für dieses Projekt, falls Sie die Beispiele nicht händisch nachvollziehen wollen.

Beispiel:

```
<software Installierbar = "Ja" >Visual Studio 2008 </software>
```

Das Element `Software` besitzt ein Attribut `Installierbar` und den Wert `Visual Studio 2008`. Elemente in XML beginnen mit einem Starttag `<name>` und enden mit dem Endtag `</name>`, wobei *name* durch ein beliebiges Wort getauscht werden kann (Achtung: keine Leerzeichen!).

Zwischen `<name>` und `</name>` steht der Wert. Ist kein Wert vorhanden, kann anstelle von `<name></name>` auch `<name/>` geschrieben werden.

Attribute werden in dem Element notiert, etwa:

```
<person name="Müller" vorname="Hans"/>
```

Dieses Beispiel definiert ein Element ohne Wert, das zudem die Attribute `name` und `vorname` beinhaltet.

Eine detaillierte Beschreibung von XML würde den Umfang dieses Buches sprengen, daher sei hier auf die gute deutsche Einführung von *SelfHtml*<sup>1</sup> bzw. auf die Seiten von *W3C*<sup>2</sup> verwiesen.

## Vergleich Visual Basic 8 und Visual Basic 9

Um unter Visual Basic 8 eine XML-Struktur zu erstellen oder zu ändern, musste einiger Aufwand betrieben werden. Als Beispiel soll hier folgende XML-Struktur um das Element `baujahr` erweitert werden:

```
<fuhrpark>  
  <kennzeichen>MS - VB 2008</kennzeichen>  
  <ladung menge="10 Tonnen">Salz</ladung>  
  <hersteller>MAN</hersteller>  
  <baujahr>1998</baujahr>  
</fuhrpark>
```

<sup>1</sup> <http://de.selfhtml.org>

<sup>2</sup> <http://www.w3c.org>

Der Code lautete hierzu für VB8:

```
Dim xml As New XmlDocument
xml.Load(Application.StartupPath + "\fuhrpark1.xml")
Dim nodeList As XmlNodeList = xml.GetElementsByTagName("fuhrpark")
For Each node As XmlNode In nodeList
    Dim xmlElement As XmlElement = xml.CreateElement("baujahr")
    xmlElement.InnerText = "1998"
    node.AppendChild(xmlElement)
Next
```

In Visual Basic 9 gibt es viele Vereinfachungen bezüglich der XML-Verarbeitung. Es sind neue Klassen hinzugekommen. Die wichtigsten sind `XDocument`, `XElement` und `XAttribute`.

Um nun die obere Struktur zu erhalten, reicht es aus, folgenden Code zu erstellen:

```
Dim xml As New XDocument
xml.Add(New XElement("fuhrpark", _
    New XElement("kennzeichen", "MS - VB 2008"), _
    New XElement("ladung", _
        New XAttribute("menge", "10 Tonnen"), _
        "Salz" _
    ), _
    New XElement("hersteller", "MAN") _
))
```

Unter dem Element `fuhrpark` werden die Elemente `kennzeichen`, `ladung` und `hersteller` gespeichert. Zudem erhält das Element `ladung` ein weiteres Attribut namens `menge`. Das Hinzufügen des `baujahr`-Elements kann nun durch folgende Zeile erreicht werden.

```
xml.Element("fuhrpark").Add(New XElement("baujahr", 1998))
```

Der Wert eines `XElement`s (z.B. 1998) kann über die `Value` Property erfragt werden.

## XML-Literale – XML direkt im Code ablegen

Mit Visual Basic 9 kann man `XDocument`- und `XElement`-Objekte direkt im Code erstellen – die entsprechenden Elemente dafür nennen sich *XML-Literale*. Mithilfe von XML-Literalen ist es möglich, XML inline direkt im Quellcode zu erstellen, wie das folgende Beispiel zeigt:

```
Dim fuhrpark As XElement = <fuhrpark>
    <kennzeichen>MS - VB 2008</kennzeichen>
    <ladung menge="10 Tonnen">Salz</ladung>
    <hersteller>MAN</hersteller>
</fuhrpark>
```

Beim fuhrpark-Objekt handelt es sich um ein gewöhnliches XElement und baujahr kann nun wie gewohnt hinzugefügt werden:

```
fuhrpark.Add(New XElement("baujahr", 1998))
```

Mit XML-Literalen geht das sogar noch einfacher:

```
fuhrpark.Add(<baujahr>1998</baujahr>)
```

Darüber hinaus ist es auch möglich, Ausdrücke mit <%= %> als XML in den Code einzubetten, wie die folgenden beiden Zeilen zeigen:

```
Dim myBaujahr As Integer = 1998
fuhrpark.Add(<baujahr><%= myBaujahr %></baujahr>)
```

Beim Aufrufen der Add-Methode wird der Wert des baujahr-Elements durch die Variable myBaujahr ersetzt.

## Erstellen von XML mithilfe von LINQ

Mithilfe von LINQ können ebenso XML-Dokumente bzw. -Elemente erstellt werden.

Aus einer bestehenden Liste, die durch die folgenden Codezeilen erstellt wird ...

```
Dim fahrzeugliste As New List(Of Fahrzeug)
fahrzeugliste.Add(New Fahrzeug With {.Kennzeichen = "MS-VB 2008", .Hersteller = "MAN", _
    .Ladung = New Fahrzeug.LadungsBeschreibung With
        {.Menge = 10000, .Gut = "Salz"}})
fahrzeugliste.Add(New Fahrzeug With {.Kennzeichen = "MS-C# 2008", .
    Hersteller = "Mercedes-Benz",
    .Ladung = New Fahrzeug.LadungsBeschreibung With
        {.Menge = 20000, .Gut = "Erdnüsse"}})
fahrzeugliste.Add(New Fahrzeug With {.Kennzeichen = "MS-J# 2008", .Hersteller = "DAF",
    .Ladung = New Fahrzeug.LadungsBeschreibung With
        {.Menge = 5000, .Gut = "Wackeldackel"}})

Private Class Fahrzeug
    Public Class LadungsBeschreibung
        Private _menge As Double
        Private _gut As String

        Public Property Menge() As Double
        ...
        End Property

        Public Property Gut() As String
        ...
        End Property
    End Class
```

```

Private _kennzeichen As String
Private _ladung As LadungsBeschreibung
Private _hersteller As String

Public Property Kennzeichen() As String
...
End Property

Public Property Ladung() As LadungsBeschreibung
...
End Property

Public Property Hersteller() As String
...
End Property
End Class

```

... soll folgendes XML-Element erstellt werden:

```

<fuhrpark>
  <fahrzeug>
    <kennzeichen>MS-VB 2008</kennzeichen>
    <hersteller>MAN</hersteller>
    <ladung menge="10000">Salz</ladung>
  </fahrzeug>
  <fahrzeug>
    <kennzeichen>MS-C# 2008</kennzeichen>
    <hersteller>Mercedes-Benz</hersteller>
    <ladung menge="20000">Erdnüsse</ladung>
  </fahrzeug>
  <fahrzeug>
    <kennzeichen>MS-J# 2008</kennzeichen>
    <hersteller>DAF</hersteller>
    <ladung menge="5000">Wackeldackel</ladung>
  </fahrzeug>
</fuhrpark>

```

Das Anlegen der XML-Strukturen erfolgt dabei mit XElement und XAttribute:

```

Dim fuhrpark = New XElement("fuhrpark", _
    From fzg In fahrzeugliste _
    Select New XElement("fahrzeug", _
        New XElement("kennzeichen", fzg.Kennzeichen), _
        New XElement("hersteller", fzg.Hersteller), _
        New XElement("ladung", fzg.Ladung.Gut,
            New XAttribute("menge", fzg.Ladung.Menge)) _
        )
    )

```

Dem XElement mit dem Namen fuhrpark wird eine Liste von XElementen (fahrzeug) hinzugefügt. Diese Liste (genaugenommen IEnumerable(of XElement)) wird mit *LINQ to Objects* erstellt. Das fuhrpark-Element wird zudem um die entsprechenden Unterelemente erweitert.

Auch hier ist wieder die Verwendung von Literalen möglich:

```
fuhrpark = <fuhrpark>
    <%= From fzg In fahrzeugliste _
        Select <fahrzeug>
            <kennzeichen><%= fzg.Kennzeichen %></kennzeichen>
            <hersteller><%= fzg.Hersteller %></hersteller>
            <ladung menge=<%= fzg.Ladung.Menge %>>
                <%= fzg.Ladung.Gut %>
            </ladung>
        </fahrzeug> _
    %>
</fuhrpark>
```

## Abfragen von XML-Dokumenten mit LINQ to XML

LINQ bietet mit *LINQ to XML* auch eine Abfrageunterstützung für XML-Strukturen an.

Ein Beispiel:

```
Dim fuhrpark = <?xml version="1.0"?>
    <fuhrpark>
        <fahrzeug>
            <kennzeichen>MS-VB 2008</kennzeichen>
            <hersteller>MAN</hersteller>
            <ladung menge="10000">Salz</ladung>
        </fahrzeug>
        <fahrzeug>
            <kennzeichen>MS-C# 2008</kennzeichen>
            <hersteller>Mercedes-Benz</hersteller>
            <ladung menge="20000">Erdnüsse</ladung>
        </fahrzeug>
        <fahrzeug>
            <kennzeichen>MS-J# 2008</kennzeichen>
            <hersteller>DAF</hersteller>
            <ladung menge="5000">Wackeldeckel</ladung>
        </fahrzeug>
    </fuhrpark>
Dim manFahrzeuge = From fahrzeug In fuhrpark...<fahrzeug> _
    Where fahrzeug.<hersteller>.Value = "MAN"
For Each fahrzeug In manFahrzeuge
    Console.WriteLine(fahrzeug)
Next
```

Bei der Variablen `fuhrpark` handelt es sich um ein `XDocument`. In der `From`-Klausel wird auf alle `XElement`s mit dem Namen `Fahrzeug` zugegriffen. Die `Where`-Klausel prüft den Hersteller des Fahrzeugs auf den Eintrag »MAN«. Der Wert eines `XElement`s muss über die `Value` Property erfragt werden.

Es werden zwei Zugriffsmöglichkeiten für Elemente bereitgestellt:

Der Zugriff auf direkte Unterelemente erfolgt über `.<Element-name>`. Zudem kann auf darunterliegende Elemente mit `...<Element-name>` zugegriffen werden, wobei sich das Element durchaus tief in den XML-Strukturen befinden darf.

Auf den Inhalt eines Attributes kann über `.@<Attribut-name>` zugegriffen werden.

Beispiel: Es sollen alle schwer beladenen Fahrzeuge ermittelt werden.

```
Dim schwerBeladen = From fahrzeug In fuhrpark...<fahrzeug> _
                    Where fahrzeug.<ladung>.@menge > 15000
For Each fahrzeug In schwerBeladen
    Console.WriteLine(fahrzeug)
Next
```

ergibt folgende Ausgabe:

```
<fahrzeug>
  <kennzeichen>MS-C# 2008</kennzeichen>
  <hersteller>Mercedes-Benz</hersteller>
  <ladung menge="20000">Erdnüsse</ladung>
</fahrzeug>
```

## IntelliSense-Unterstützung für LINQ to XML-Abfragen

Bei der Arbeit mit XML-Daten ist jede Unterstützung der IDE sehr hilfreich. Visual Basic 2008 bietet genau diese an, wie in der folgenden Abbildung zu sehen:

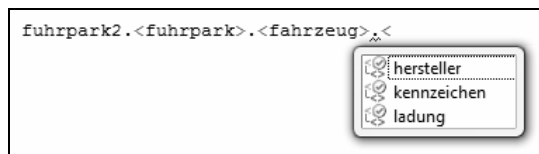


Abbildung 10.1 XML IntelliSense-Unterstützung

Diese Unterstützung erhält man, sobald eine XML Schema Definition (XSD)-Datei mit Imports in die Codedatei importiert wird.

Aber fangen wir mal vorne an. Eine XSD-Datei kann Visual Studio selbst erstellen. Hierzu erstellt man zunächst die komplette XML-Datei mit allen möglichen Ausprägungen.

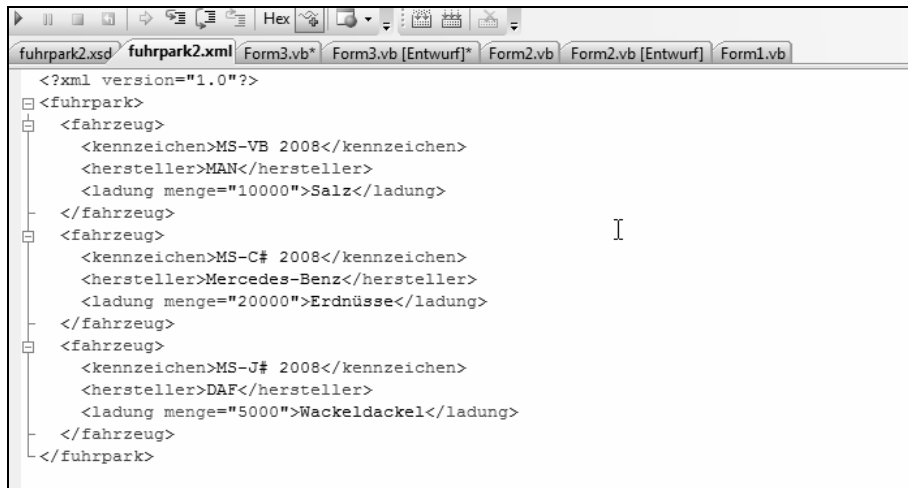


Abbildung 10.2 Eine XML-Datei in Visual Studio

Danach kann man mithilfe des gleichlautenden Befehls im Menü *XML* ein Schema erstellen:

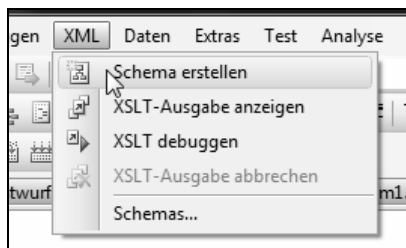


Abbildung 10.3 Erstellen einer XML Schema Definition (XSD)

Das erstellte Schema muss nun im Projektverzeichnis gespeichert und in das Projekt mitaufgenommen werden.

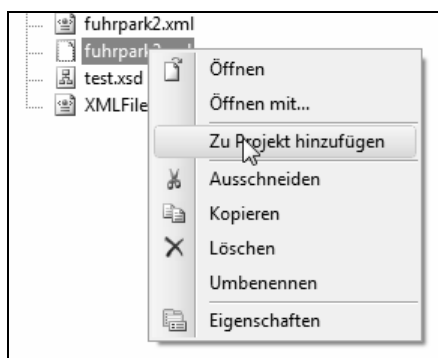


Abbildung 10.4 Hinzufügen einer Datei im Projektmappen-Explorer

In der soeben erstellten XSD-Datei muss nun noch ein Ziel-Namensbereich angegeben werden. Dieser Namensbereich muss nicht existieren, sondern dient lediglich der Klassifikation. In diesem Fall wurde `http://mysample.local.de/fuhrpark2` verwendet.

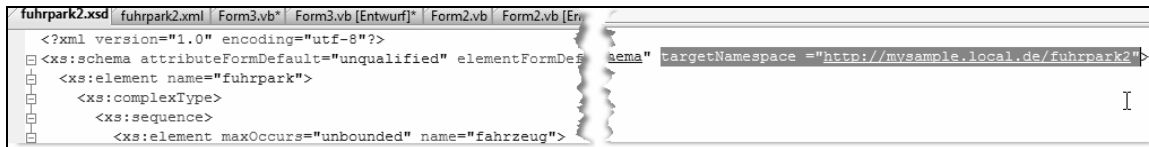


Abbildung 10.5 Erweitern einer XSD-Datei um den Ziel-Namensbereich

In der VB Datei, die eine XML-Unterstützung erhalten soll, kann jetzt der soeben festgelegte Ziel-Namensbereich importiert werden.

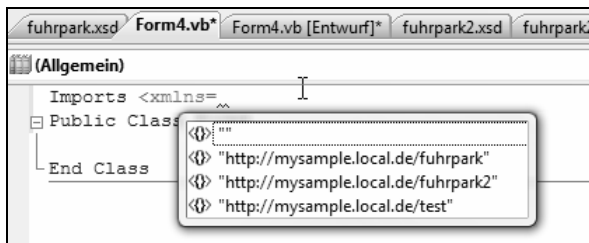


Abbildung 10.6 Hinzufügen einer Imports Anweisung für XML Schema Dateien

Werden mehrere Namensräume verwendet, bietet es sich an, den Namensraum mit einem Präfix zu versehen. Das Präfix wird nach `xmlns` notiert und mit einem Doppelpunkt von `xmlns` getrennt. In diesem Fall wird als Präfix `fuhrpark` verwendet. Bei nur einem Namensraum, der in die VB-Coddatei importiert wird, ist das nicht erforderlich.

```
Imports <xmlns:fuhrpark="http://mysample.local.de/fuhrpark2">
Imports <xmlns:artikel="http://mysample.local.de/artikel">
```

Die XML-Strukturen müssen in diesem Fall auch um die Präfixe erweitert werden.

```
Dim fuhrpark = <fuhrpark:fuhrpark>
    <fuhrpark:fahrzeug>
        <fuhrpark:kennzeichen>MS-VB 2008</fuhrpark:kennzeichen>
        <fuhrpark:hersteller>MAN</fuhrpark:hersteller>
        <fuhrpark:ladung menge="10000">Salz</fuhrpark:ladung>
    </fuhrpark:fahrzeug>
</fuhrpark:fuhrpark>
Dim fuhrpark = <fuhrpark:fuhrpark>
    <fuhrpark:fahrzeug>
        <fuhrpark:kennzeichen>MS-C# 2008</fuhrpark:kennzeichen>
        <fuhrpark:hersteller>Mercedes-Benz</fuhrpark:hersteller>
        <fuhrpark:ladung menge="20000">Erdnüsse</fuhrpark:ladung>
    </fuhrpark:fahrzeug>
</fuhrpark:fuhrpark>
```

```
<fuhrpark:kennzeichen>MS-J# 2008</fuhrpark:kennzeichen>  
<fuhrpark:hersteller>DAF</fuhrpark:hersteller>  
<fuhrpark:ladung menge="5000">Wackel dackel</fuhrpark:ladung>  
</fuhrpark:fahrzeug>  
</fuhrpark:fuhrpark>
```

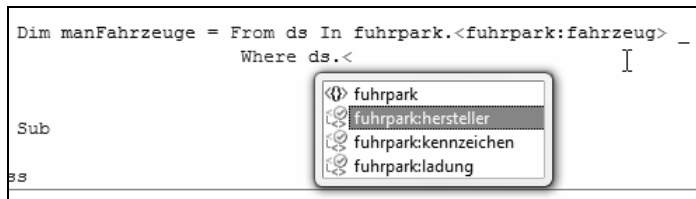


Abbildung 10.7 XML-Unterstützung der IDE