

---

# 01 LINQ

## Language Integrated Query

---

# Agenda

---

- Anonyme Datentypen
- Einführung in LINQ-to-Objects
- LINQ-Abfrageoperationen
- LINQ-Methoden und -Operatoren
- Lambda-Ausdrücke
- XML
- LINQ-to-XML
- LINQ-to-SQL

## Anonyme Datentypen

- Mit anonymen Typen können schreibgeschützte Eigenschaften in ein einzelnes Objekt gekapselt werden, ohne zuerst einen Typ explizit definieren zu müssen.
- Der Typname wird vom Compiler generiert und ist nicht auf Quellcodeebene verfügbar.
- Der Typ der Eigenschaften wird vom Compiler abgeleitet.

# Anonyme Datentypen

---

## Anonyme Datentypen

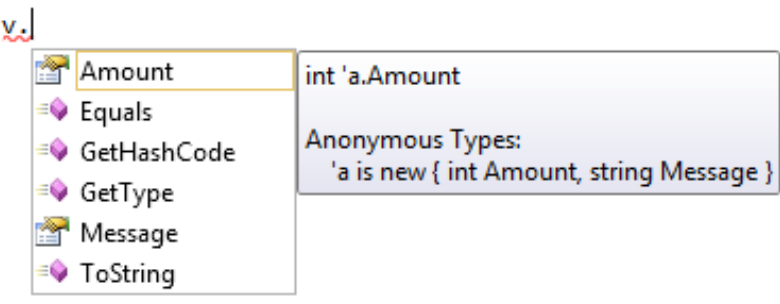
- Sind Referenztypen, die sich direkt vom Objekt ableiten.
- Der Compiler gibt ihnen einen Namen. Die Anwendung kann nicht darauf zugreifen.
- Für die CLR unterscheidet sich ein anonymer Typ nicht von anderen Referenztypen. Er kann jedoch in keinen anderen Typ als object umgewandelt werden.

# Anonyme Datentypen

## Anonyme Datentypen

- Anonyme Typen werden normalerweise in der `select`-Klausel eines Abfrageausdrucks in LINQ verwendet, um einen Untersatz der Eigenschaften aus jedem Objekt in der Quellsequenz zurückzugeben.
- Anonyme Typen werden mithilfe des Operators `new` mit einem Objektinitialisierer erstellt.
- Beispiel:

```
class Program
{
    static void Main(string[] args)
    {
        var v = new { Amount = 108, Message = "Hello" };
    }
}
```



## Anonyme Datentypen

- Anonyme Typen sind Klassen-Typen, die aus einer oder mehreren öffentlichen, schreibgeschützten Eigenschaften bestehen.
- Es werden keine anderen Arten von Klassen-Memberrn, z.B. Methoden oder Ereignisse, zugelassen.
- Ein anonymer Typ kann in keine Schnittstelle und keinen Typ mit Ausnahme von `object` umgewandelt werden.

# Anonyme Datentypen

---

## Anonyme Datentypen

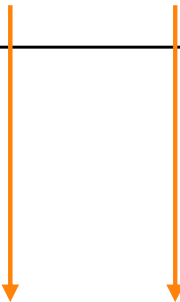
### ■ Das häufigste Szenario:

- Einen anonymen Typ mit einigen Eigenschaften von einem anderen Typ zu initialisieren.

### ■ Beispiel:

- Eine Klasse mit dem Namen Product, mit Color-Eigenschaft und Price-Eigenschaft sowie mehreren anderen Eigenschaften.
- Products ist eine Auflistung von Product-Objekten.
- Bei der Abfrage interessieren uns nur Color und Price.
- Keine Namen für Eigenschaften definiert → Namen werden übernommen

```
var productQuery =  
    from prod in products  
    select new { prod.Color, prod.Price };  
  
foreach (var v in productQuery)  
{  
    Console.WriteLine("Color={0}, Price={1}", v.Color, v.Price);  
}
```



## LINQ-Provider

- LINQ-to-Objects
  - Fundament aller LINQ-Abfragen, damit lassen sich Auflistungen und Objekte manipulieren.
- LINQ-to-XML
  - Programmierschnittstelle für XML im Arbeitsspeicher.
- LINQ-to-SQL
  - LINQ-Provider für SQL Server 2005/2008.
- LINQ-to-ADO.NET
  - LINQ-to-DataSet: Umfangreichere Abfragen auf DataSets
  - LINQ-to-SQL: Abfragen auf SQL-Server-Datenbankschemas.
- Namensraum: `System.Linq`



## Einführung in LINQ

- LINQ-Abfrageausdrücke werden mit einer deklarativen Abfragesyntax geschrieben (ab C# 3.5)
- Mit einer Abfragesyntax können komplexe Filter-, Sortier- und Gruppieroperationen mit minimalem Code für Datenquellen durchgeführt werden.
- Es werden dieselben grundlegenden Abfrageausdrucksmuster zum Abfragen und Transformieren von Daten SQL-Datenbanken, ADO.NET-Datasets, XML-Dokumenten und -Streams sowie .NET-Auflistungen verwendet.

## Einführung in LINQ

- Eine LINQ-Abfrage wird meist auf Daten in Form einer Liste angewandt und es wird eine neue Liste erzeugt. Diese muss nicht vom gleichen Typ sein.
- LINQ basiert auf folgenden Konzepten:
  - Implizite Typzuweisung
  - Anonyme Datentypen
  - Objektserialisierung
  - Die resultierende Liste hat für den Compiler einen klaren Typ.

```
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };  
  
var lowNums =  
    from n in numbers  
    where n < 5  
    select n;
```

# LINQ-Abfrageoperationen

---

## Eine Abfrageoperation besteht aus drei Teilen (1/2)

- Datenquelle
- Abfrageausdruck
- Ausführen des Abfrageausdrucks
- Codebeispiel: Alle Scores über 80 ermitteln

```
// Datenquelle
int[] scores = new int[] { 32, 99, 91, 85, 50, 80 };

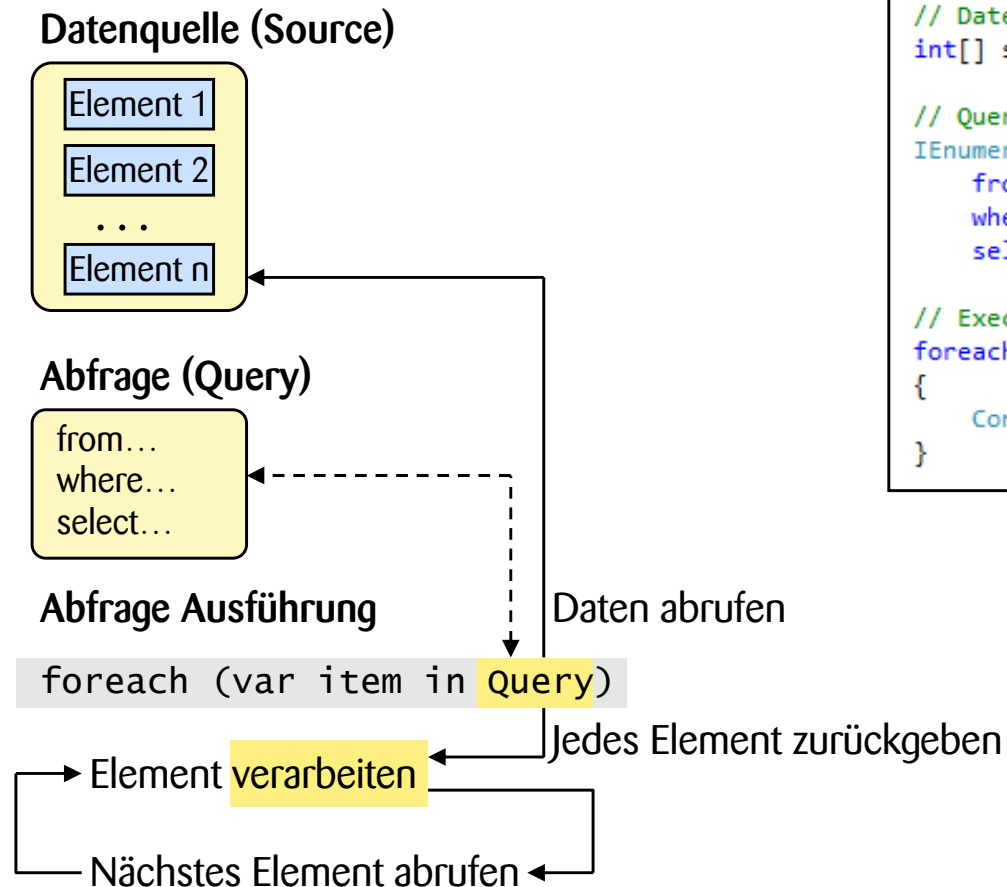
// Query Expression - Abfrageausdruck
IEnumerable<int> scoreQuery =
    from score in scores
    where score > 80
    select score;

// Execute the query - Abfrageausdruck ausführen
foreach (int i in scoreQuery)
{
    Console.WriteLine(i + " ");    // Ausgabe: 99 91 85
}
```

# LINQ-Abfrageoperationen

## Eine Abfrageoperation besteht aus drei Teilen (2/2)

### ■ Ablauf einer Abfrageoperation



```
// Datenquelle
int[] scores = new int[] { 32, 99, 91, 85, 50, 80 };

// Query Expression - Abfrageausdruck
IEnumerable<int> scoreQuery =
    from score in scores
    where score > 80
    select score;

// Execute the query - Abfrageausdruck ausführen
foreach (int i in scoreQuery)
{
    Console.WriteLine(i + " ");    // Ausgabe: 99 91 85
}
```

## Abfrageausdruck

### ■ Quelldaten

- Die Quelldaten werden immer als eine `IEnumerable<T>`-Auflistung oder `IQueryable<T>`-Auflistung interpretiert.
- In LINQ-to-XML werden die Quelldaten als `IEnumerable<XElement>` sichtbar gemacht.
- In LINQ-to-DataSet ist es ein `IEnumerable<DataRow>`.
- In LINQ-to-SQL ist es ein `IEnumerable` oder `IQueryable` eines beliebigen benutzerdefinierten Objekts.

### ■ Abfrage Beispiel 1:

```
IEnumerable<int> scoreQuery =  
    from score in scores  
    where score > 80  
    select score;
```

# LINQ-Abfrageoperationen

---

## Abfrageausdruck

### ■ Beispiel 2: Resultat hat anderen Datentyp

```
// Anderer Datentyp als Resultat
IEnumerable<string> highScoresQuery2 =
    from score in scores
    where score > 80
    orderby score descending
    select String.Format("The score is {0}", score);
```

### ■ Beispiel 3: Einzelner Wert als Resultat

```
// Einzelner Wert abfragen
int highScoreCount =
    (from score in scores
     where score > 80
     select score)
    .Count();
```

## Abfrageausdruck

- Besteht aus einem Klauselsatz (ähnlich SQL) und muss mit einer `from`-Klausel beginnen und mit einer `select`-Klausel oder einer `group`-Klausel enden.
- Zwischen der ersten `from`-Klausel und der letzten `select`-Klausel bzw. `group`-Klausel, können sich eine oder mehrere der folgenden optionalen Klauseln befinden: `where`, `orderby`, `join`, `let` sowie die zusätzlichen `from`-Klauseln.
- Das Schlüsselwort `into` kann verwendet werden, um das Ergebnis einer `join`- oder `group`-Klausel oder `group`-Klausel zu aktivieren, um als Quelle für zusätzliche Abfrageklauseln im gleichen Abfrageausdruck zu fungieren.

### Abfrage (Query)

`from...`  
`where...`  
`select...`

# LINQ-Methoden und -Operatoren

## Daten sortieren

### ■ Methoden:

- OrderBy, OrderByDescending, ThenBy, ThenByDescending, Reverse

### ■ Beispiel: Wörter nach Länge sortieren

```
string[] words = { "Mit", "LINQ", "rockt", "der", "C#", "Code" };

IEnumerable<string> query = from word in words
                           orderby word.Length
                           select word;

foreach (string str in query)
    Console.WriteLine(str);

/* Ausgabe:
    C#
    Mit
    der
    LINQ
    Code
    rockt
*/
```

Quelle

C B D A E

Resultat

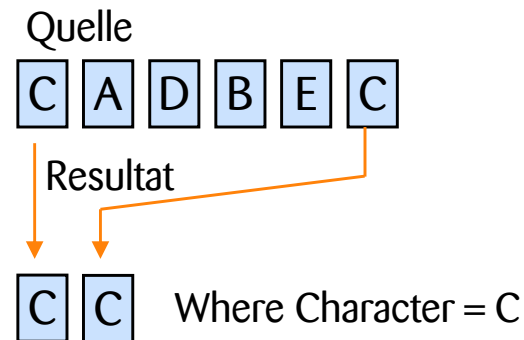
A B C D E



# LINQ-Methoden und -Operatoren

## Daten filtern

- Methoden:
  - TypeOf, Where



- Beispiel: Wörter filtern nach Länge = 4

```
string[] words = { "Mit", "LINQ", "rockt", "der", "C#", "Code" };

IEnumerable<string> query = from word in words
                           where word.Length == 4
                           select word;

foreach (string str in query)
    Console.WriteLine(str);

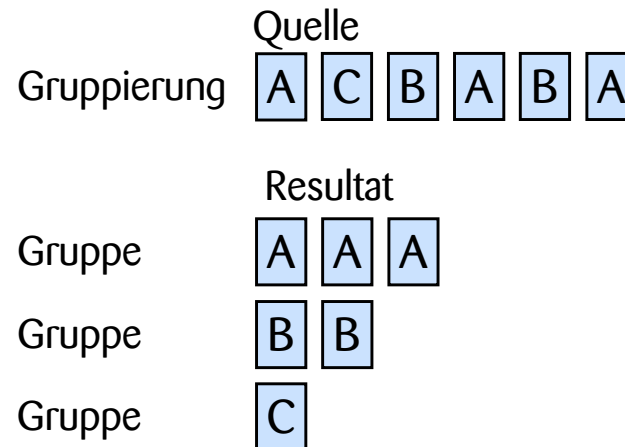
/* Ausgabe:
    LINQ
    Code
*/
```

# LINQ-Methoden und -Operatoren

## Daten gruppieren

### ■ Methoden:

- GroupBy, ToLookup



### ■ Beispiel: Gerade und ungerade Zahlen gruppieren

Ausgabe:

```
List<int> numbers = new List<int>() { 56, 25, 65, 10, 64, 52313, 3, 216 };

IEnumerable<IGrouping<int, int>> query = from number in numbers
                                         group number by number % 2;

foreach (var group in query)
{
    Console.WriteLine(group.Key == 0 ? "\nGerade Zahlen:" : "\nUngerade Zahlen:");
    foreach (int inx in group)
        Console.WriteLine(inx);
}
```

Gerade Zahlen:

56  
10  
64  
216

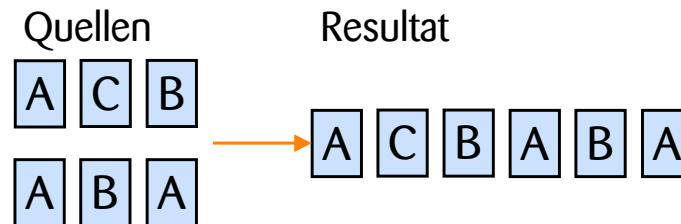
Ungerade Zahlen:

25  
65  
52313  
3

# LINQ-Methoden und -Operatoren

## Daten verketteten

- Methoden:
  - Concat



- Beispiel:

```
string[] text1 = { "Mit", "LINQ", "rockt", "der", "C#", "Code" };
string[] text2 = { "und", "er", "wird", "schlanker" };

IEnumerable<string> concatQuery =
    text1.Concat(text2).OrderBy(s => s);

foreach (string word in concatQuery)
{
    Console.WriteLine(word);
}
```

Ausgabe:

```
/* Ausgabe:
C#
Code
der
er
LINQ
Mit
rockt
schlanker
und
wird
*/
```

# Übung 1.1



## Mitarbeiterdaten abfragen (1/2)

- Erstelle eine Applikation (Konsole oder Windows Forms) zur Abfrage von Mitarbeiterdaten mit LINQ
- A) Erstelle eine Klasse Mitarbeiter mit Attributen:
  - Personalnummer, Name, Vorname, Abteilung, Strasse, PLZ, Wohnort, Telefon, Salär
- B) Erstelle eine Mitarbeiterliste und füge min. 10 Mitarbeiter hinzu.
  - Tipp: Verwende die Kurzform zur Instanziierung der Mitarbeiter

# Übung 1.1



## Mitarbeiterdaten abfragen (2/2)

- C) Erstelle die folgenden Abfragen mit LINQ und visualisiere sie auf dem Bildschirm:
  - C.1) Alle Mitarbeiter, sortiert nach Nachname
  - C.2) Alle Mitarbeiter, gruppiert nach Abteilung
  - C.3) Alle Mitarbeiter deren Salär > CHF 100'000
  - C.4) Alle Wohnorte mit PLZ = 8xxx (mit 8 beginnend und vierstellig)
  
- D) Optional: Letzte Prüfungsaufgaben nehmen und schauen, wo man eine Schleife mit einer LINQ-Abfrage ersetzen könnte

# Abfrage-Ausdrücke

## Cross-Joins

- Operator: Join
- Beispiel:

```
string[] categories = new string[]{  
    "Beverages",  
    "Condiments",  
    "Vegetables",  
    "Dairy Products",  
    "Seafood" };  
  
List<Product> products = GetProductList();  
  
var q =  
    from c in categories  
    join p in products on c equals p.Category  
    select new { Category = c, p.ProductName };  
  
foreach (var v in q)  
{  
    Console.WriteLine(v.ProductName + ": " + v.Category);  
}
```

```
public class Product  
{  
    public int ProductID { get; set; }  
    public string ProductName { get; set; }  
    public string Category { get; set; }  
    public decimal UnitPrice { get; set; }  
    public int UnitsInStock { get; set; }  
}
```

Ausgabe:

```
Camembert Pierrot: Dairy Products  
Gudbrandsdalsost: Dairy Products  
Flotemysost: Dairy Products  
Mozzarella di Giovanni: Dairy Products  
Ikura: Seafood  
Konbu: Seafood  
Carnarvon Tigers: Seafood
```

sbildung

# Abfrage-Ausdrücke

## Group-Joins

### ■ Beispiel:

```
string[] categories = new string[]{
    "Beverages",
    "Condiments",
    "Vegetables",
    "Dairy Products",
    "Seafood" };

List<Product> products = GetProductList();

var q =
    from c in categories
    join p in products on c equals p.Category into ps
    select new { Category = c, Products = ps };

foreach (var v in q)
{
    Console.WriteLine(v.Category + ":");
    foreach (var p in v.Products)
    {
        Console.WriteLine("    " + p.ProductName);
    }
}
```

### Ausgabe:

Dairy Products:  
Queso Cabrales  
Queso Manchego La Pastora  
Gorgonzola Telino  
Mascarpone Fabioli  
Geitost  
Raclette Courdavault  
Camembert Pierrot  
Gudbrandsdalsost  
Flotemysost  
Mozzarella di Giovanni  
Seafood:  
Ikura  
Konbu  
Carnarvon Tigers  
Nord-Ost Matjeshering

# CSV-Dateien mit LINQ verarbeiten

## CSV-Datei

- Mit LINQ können Datensätze aus einer CSV-Datei einfach selektiert werden
- Beispiel: Alle Bücher von Autor Jack Bauer mit Preis unter CHF 30.- selektieren

### CSV-Datei: Books.csv

```
Id;Author;Title;Genre;Price;PublishDate;Description
1;Jack Bauer;Season 1;Thriller;18.50;2004-12-21;Jack is hunting for bad guys.
2;Jack Bauer;Season 2;Thriller;23.50;2005-12-21;Jack is hunting again for bad guys.
3;Jack Bauer;Season 3;Thriller;25.50;2006-12-21;Jack is hunting again for bad guys.
4;Steven King;The Shining;Thriller;14.50;1990-11-21;A bad guy in a hotel.
5;Jack Bauer;Season 4;Thriller;28.50;2007-12-21;Jack is hunting again for bad guys.
6;Jack Bauer;Season 5;Thriller;32.50;2008-12-21;Jack is hunting again for bad guys.
7;Tolkien;The Hobbit;Adventure;19.50;2010-12-21;An unexpected journey.
8;Jack Bauer;Season 6;Thriller;39.50;2009-12-21;Jack is hunting again for bad guys.
```



# CSV-Dateien mit LINQ verarbeiten

CSV-Datei: Books.csv

## Beispielcode:

```
string delimiter = ",";
List<string> lines = File.ReadAllLines(@"..\..\Books.csv").ToList<string>();

List<string> JackBauersCheapBooks = lines
    // Leere Zeilen ignorieren
    .Where(line => !string.IsNullOrEmpty(line))
    |
    // Zeile splitten mit Delimiter
    .Select(line => line.Split(delimiter.ToCharArray(), StringSplitOptions.RemoveEmptyEntries))

    // Feld 2 vergleichen, Alle Buecher von Jack Bauer selektieren
    .Where(values => values[1].Equals("Jack Bauer", StringComparison.CurrentCultureIgnoreCase))

    //// Feld 5 vergleichen, alle Zeilen mit Preis <= 30 selektieren
    .Where(values => double.Parse(values[4]) <= 30)

    // Gesplittete Felder wieder zusammen fügen mit join
    .Select(values => string.Join(" - ", values))

    .ToList<string>();// Zu Liste konvertieren

foreach (string text in JackBauersCheapBooks)
{
    Console.WriteLine(text);
}

// Ausgabe:
// 1 - Jack Bauer - Season 1 - Thriller - 18.50 - 2004-12-21 - Jack is hunting for bad guys.
// 2 - Jack Bauer - Season 2 - Thriller - 23.50 - 2005-12-21 - Jack is hunting again for bad guys.
// 3 - Jack Bauer - Season 3 - Thriller - 25.50 - 2006-12-21 - Jack is hunting again for bad guys.
// 5 - Jack Bauer - Season 4 - Thriller - 28.50 - 2007-12-21 - Jack is hunting again for bad guys.
```

```
Id;Author;Title;Genre;Price;PublishDate;Description
1;Jack Bauer;Season 1;Thriller;18.50;2004-12-21;Jack is hunting for bad guys.
2;Jack Bauer;Season 2;Thriller;23.50;2005-12-21;Jack is hunting again for bad guys.
3;Jack Bauer;Season 3;Thriller;25.50;2006-12-21;Jack is hunting again for bad guys.
4;Steven King;The Shining;Thriller;14.50;1990-11-21;A bad guy in a hotel.
5;Jack Bauer;Season 4;Thriller;28.50;2007-12-21;Jack is hunting again for bad guys.
6;Jack Bauer;Season 5;Thriller;32.50;2008-12-21;Jack is hunting again for bad guys.
7;Tolkien;The Hobbit;Adventure;19.50;2010-12-21;An unexpected journey.
8;Jack Bauer;Season 6;Thriller;39.50;2009-12-21;Jack is hunting again for bad guys.
```

# Lambda-Ausdrücke

## Grundlagen

### ■ Was ist ein Lambda-Ausdruck?

- Ein Lambda-Ausdruck ist eine anonyme Funktion/Methode, die Ausdrücke und Anweisungen enthalten kann

### ■ Lambda-Operator =>

- Alle Lambda-Ausdrücke verwenden den Operator Lambda =>, der so viel bedeutet wie "geht über in".
- Auf der linken Seite des Operators werden die Eingabeparameter angegeben (falls vorhanden), und auf der rechten Seite befindet sich der Ausdruck oder Anweisungsblock.

Beispiel: `x => x * x` bedeutet "x geht über in x mal x".

### ■ Explizit/implizite Parameterübergabe

```
(int x) => x + 1 // explicitly typed parameter
(y,z) => y * z  // implicitly typed parameter
```

# Lambda-Ausdrücke

## Grundlagen

- Beispiel: Gerade Zahlen mit Lambda-Ausdruck aus Liste ermitteln

```
static void Main(string[] args)
{
    List<int> zahlen = new List<int>() {1,2,3,4,5,6,7,8,9,10};
    LambdaExampleGetEvenNo(zahlen);
}

static void LambdaExampleGetEvenNo(List<int> list)
{
    // Alle geraden Zahlen ermitteln
    var evenNumbers = list.FindAll(i => (i % 2) == 0);
    foreach (int i in evenNumbers)
    {
        Console.WriteLine(i);
    }
}
```

↑  
Lambda-Ausdruck

FindAll ist nicht LINQ, sondern eine Methode auf List<T>.

LINQ wäre Enumerable.Where.

# Übung 1.2



## Mitarbeiterdaten mit Lambda-Ausdrücken abfragen

- A) Ergänze die Übung 1.1 mit Abfragen, die Lambda-Ausdrücke verwenden:
  - A.1) Alle Mitarbeiter, derer Salär  $>$  CHF 100'000
  - A.2) Alle Wohnorte, derer Länge  $<$  6

## Grundlagen

### ■ Standard

- XML ist ein W3C-Standard

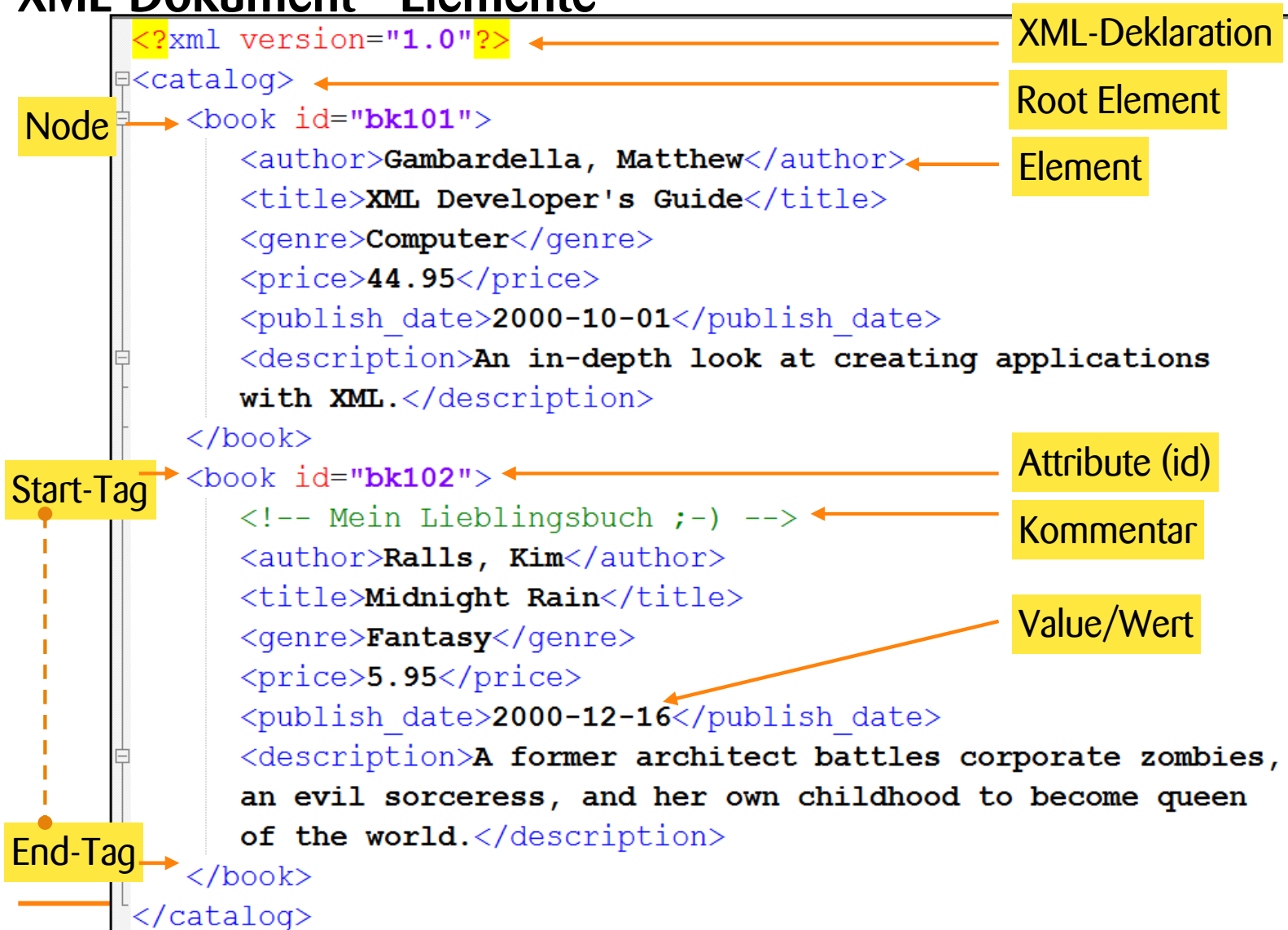
### ■ Begriffe

- XML: Extended Markup Language
- XSD: XML Schema Definition, Schema beschreibt XML Dokument Struktur und Datentypen
- XPATH: XML Query Language (Abfragesprache)
- XSLT: XML Transformation
- SAX: Simple API for XML
- DOM: Document Object Model
- XML-Parser: Es gibt XML-Parser zum Lesen von XML in nahezu allen Programmiersprachen

## XML-Dokument-Beispiel

```
<?xml version="1.0"?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications
      with XML.</description>
  </book>
  <book id="bk102">
    <!-- Mein Lieblingsbuch ;-) -->
    <author>Ralls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate zombies,
      an evil sorceress, and her own childhood to become queen
      of the world.</description>
  </book>
</catalog>
```

## XML-Dokument - Elemente



## Grundlagen

### ■ Sonderzeichen

| Zeichen: | Codierung in XML |
|----------|------------------|
| &        | &amp;            |
| <        | &lt;             |
| >        | &gt;             |
| “        | &quot;           |
| '        | &apos;           |

### ■ Beispiel: T&T als Wert

```
<product>T&T</product>
```

Fehlerhaft codiert

```
<product>T&amp;T</product>
```

Korrekt codiert



## Anwendungsbeispiel: Konfigurationsdatei app.config

```
<?xml version="1.0"?>
<configuration>
  <appSettings>
    <add key="ConnString" value="Provider=Microsoft.Jet.OLEDB.4.0;
      Data Source=C:\Users\sab\Documents\60 HFU\2010\10 Exercises\13 ADO.NET\PersonenVerwaltung.mdb;
      Persist Security Info=False"/>
  </appSettings>
</configuration>
```

## Code zum Lesen des Connection-Strings

```
private string GetConnString()
{
    // Zudem braucht es eine Referenz auf System.Configuration
    return ConfigurationManager.AppSettings["ConnString"];
}
```

## XML im .NET Framework

- Namensraum: `System.Xml`
- Einige wichtige Klassen:

| Klasse:                     | Beschreibung   |
|-----------------------------|--|
| <code>XmlComment</code>     | Stellt einen Xml Kommentar dar.  |
| <code>XmlDeclaration</code> | Stellt den Knoten für die XML-Deklaration <code>&lt;?xml version='1.0' ...?&gt;</code> dar.  |
| <code>XmlDocument</code>    | Implementiert das W3C-DOM, Strukturdarstellung eines XML-Dokuments im Speicher (Cache). Klasse ermöglicht das Navigieren im Dokument und dessen Bearbeitung. |
| <code>XmlTextReader</code>  | Stellt einen Reader dar, der schnellen, nicht zwischengespeicherten Vorwärtszugriff auf XML-Daten bietet.  |
| <code>XmlTextWriter</code>  | Stellt einen Writer für die schnelle, nicht zwischengespeicherte Vorwärtsgenerierung von Streams oder Dateien für XML Daten dar.                             |
| <code>XmlNode</code>        | Stellt einen einzelnen Knoten im XML-Dokument dar.   |
| <code>XmlElement</code>     | Stellt ein XML Element dar.  |
| <code>XmlAttribute</code>   | Stellt ein XML Attribut dar.   |

## XML-Daten lesen mit XmlDocument

### Codebeispiel:

```
XmlDocument doc = new XmlDocument();  
doc.Load(@"..\..\..\books.xml");  
XmlNodeList books = doc.SelectNodes("//catalog/book");  
  
foreach (XmlNode book in books)  
{  
    XmlNode title = book.SelectSingleNode("title");  
    XmlNode price = book.SelectSingleNode("price");  
    Console.WriteLine("Book Title: {0}, Price: {1}", title.InnerText, price.InnerText);  
}
```

Lesen mit Methode Load (...)

Nodes selektieren mit XPath-Ausdruck

Elemente selektieren mit XPath-Ausdruck

### Ausgabe:

```
Book Title: XML Developer's Guide, Price: 44.95  
Book Title: Midnight Rain, Price: 15.95  
Book Title: Maeve Ascendant, Price: 20.95  
Book Title: Oberon's Legacy, Price: 5.95  
Book Title: The Sundered Grail, Price: 7.95  
Book Title: Lover Birds, Price: 4.95  
Book Title: Splish Splash, Price: 18.95  
Book Title: Creepy Crawlies, Price: 24.95  
Book Title: Paradox Lost, Price: 30.95  
Book Title: Microsoft .NET: The Programming Bible, Price: 36.95  
Book Title: MSXML3: A Comprehensive Guide, Price: 17.95  
Book Title: Visual Studio 7: A Comprehensive Guide, Price: 49.95
```

## XML-Daten schreiben mit XmlDocument

### Codebeispiel:

```
XmlDocument doc = new XmlDocument();  
doc.Load(@"..\..\..\books.xml");  
XmlNode firstBook = doc.SelectSingleNode("//catalog/book");  
firstBook.Attributes["id"].InnerText = "MyBook";  
XmlNode price = doc.SelectSingleNode("//catalog/book/price");  
price.InnerText = "19.95";  
doc.Save(@"..\..\..\mybooks.xml");
```

Lesen mit Methode Load (...)

Elemente/Attribute selektieren  
mit XPath-Ausdruck  
und Werte setzen

Speichern mit Methode Save (...)

### Datei mybooks.xml:

Neue ID

```
<?xml version="1.0"?>  
<catalog>  
  <book id="MyBook">  
    <author>Gambardella, Matthew</author>  
    <title>XML Developer's Guide</title>  
    <genre>Computer</genre>  
    <price>19.95</price>  
    <publish_date>2000-10-01</publish_date>  
    <description>An in-depth look at creating applications  
      with XML.</description>  
  </book>
```

Neuer Price

## Grundlagen

- Namensraum: `System.Xml.Linq`
- Einige wichtige Klassen:

| Klasse:      | Beschreibung                                       |
|--------------|--|
| XComment     | Stellt ein XML Kommentar dar.                      |
| XDeclaration | Stellt eine XML Deklaration dar.                   |
| XDocument    | Stellt ein XML-Dokument dar.                       |
| XNode        | Stellt einen einzelnen Knoten im XML-Dokument dar. |
| XElement     | Stellt ein XML Element dar.                        |
| XAttribute   | Stellt ein XML Attribut dar.                       |

## Grundlagen

- Einfaches Abfragen und Erstellen von XML-Dokumenten mittels Methoden der Klasse `Extensions`

| Methode:                          | Beschreibung   |
|-----------------------------------|--|
| <code>Ancestors()</code>          | Liefert eine Auflistung der übergeordneten Knoten                            |
| <code>AncestorsAndSelf()</code>   | Liefert eine Auflistung mit Quellen Elemente und deren übergeordneten Knoten |
| <code>Attributes()</code>         | Liefert eine Auflistung mit allen Attributen                                 |
| <code>DescendantNodes()</code>    | Liefert eine Auflistung der Nachfolgerknoten                                 |
| <code>DescendantsAndSelf()</code> | Liefert eine Auflistung der Nachfolgerknoten inkl. Quellen Elemente          |
| <code>Elements()</code>           | Liefert eine Auflistung der untergeordneten Elementen                        |
| <code>InDocumentOrder()</code>    | Gibt eine in Dokumentreihenfolge sortierte Auflistung von Knoten zurück      |
| <code>Nodes()</code>              | Gibt eine Auflistung der untergeordneten Knoten zurück.                      |
| <code>Remove()</code>             | Löscht Knoten oder Attribute   |

## XML-Daten lesen mit XDocument

### Codebeispiel

```
XDocument books = XDocument.Load(@"..\..\books.xml");  
  
var res = from book in books.Descendants("book")  
         where (double)book.Element("price") < 20  
         select (string)book.Element("title") + " " + (string)book.Element("price");  
  
foreach (string elem in res)  
    Console.WriteLine("Book: {0}", elem);
```

Lesen mit Methode Load (...)

Daten selektieren mit  
LINQ-Ausdruck

### Ausgabe:

```
Book: Midnight Rain 15.95  
Book: Oberon's Legacy 5.95  
Book: The Sundered Grail 7.95  
Book: Lover Birds 4.95  
Book: Splish Splash 18.95  
Book: MSXML3: A Comprehensive Guide 17.95
```

```
<?xml version="1.0"?>  
<catalog>  
  <book id="bk101">  
    <author>Gambardella, Matthew</author>  
    <title>XML Developer's Guide</title>  
    <genre>Computer</genre>  
    <price>44.95</price>  
    <publish_date>2000-10-01</publish_date>  
    <description>An in-depth look at creating applications  
    with XML.</description>  
  </book>  
  <book id="bk102">  
    <author>Ralls, Kim</author>  
    <title>Midnight Rain</title>  
    <genre>Fantasy</genre>  
    <price>15.95</price>  
    <publish_date>2000-12-16</publish_date>  
    <description>A former architect battles corporate zombies,  
    an evil sorceress, and her own childhood to become queen  
    of the world.</description>
```

```
<?xml version="1.0"?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications
    with XML.</description>
  </book>
```

## XML-Daten lesen und in Objekte wandeln

### Beispiel Code:

```
var provider = CultureInfo.InvariantCulture;
var books = XDocument.Load(@"InputData\Books.xml");
```

```
var bookList =
    from book in books.Descendants("book")
    select new Book
    {
        Id = book.Attribute("id")?.Value,
        Author = book.Element("author")?.Value,
        Title = book.Element("title")?.Value,
        Genre = book.Element("genre")?.Value,
        Price = double.Parse(book.Element("price")?.Value),
        PublishDate = DateTime.ParseExact(book.Element("publish_date")?.Value, format: "yyyy-MM-dd", provider),
        Description = book.Element("description")?.Value
    };

```

Neues Book-Objekt instanziiieren

```
public class Book
{
    public string Id { get; set; }
    public string Author { get; set; }
    public string Title { get; set; }
    public string Genre { get; set; }
    public double Price { get; set; }
    public DateTime PublishDate { get; set; }
    public string Description { get; set; }
}
```

```
foreach (var book in bookList)
{
    Console.WriteLine(
        $"Book: {book.Id}, {book.Title}, {book.Author}, {book.Price} {book.PublishDate.ToShortDateString()}");
}
```

### Ausgabe:

```
Book: bk101, XML Developer's Guide, Gambardella, Matthew, 44.95, 01.10.2000
Book: bk102, Midnight Rain, Ralls, Kim, 15.95, 16.12.2000
Book: bk103, Maeve Ascendant, Corets, Eva, 20.95, 17.11.2000
```



## XML-Daten schreiben

Datei newbooks.xml:

```
<catalog>
  <book id="Twentyfour01">
    <author>Jack Bauer</author>
    <title>24 Hours</title>
    <genre>Thriller</genre>
    <price>28.50</price>
    <publish_date>2004-12-22</publish_date>
    <description>Jack is hunting for bad guys.</description>
  </book>
  <book id="Moonwalker01">
    <author>Michael Jackson</author>
    <title>24 Hours on the Moon</title>
    <genre>Thriller</genre>
    <price>59.90</price>
    <publish_date>2000-02-08</publish_date>
    <description>Micheal is taking a walk on the moon.</description>
  </book>
</catalog>
```

Codebeispiel:

```
XElement books = new XElement("catalog",
    new XElement("book",
        new XAttribute("id", "Twentyfour01"),
        new XElement("author", "Jack Bauer"),
        new XElement("title", "24 Hours"),
        new XElement("genre", "Thriller"),
        new XElement("price", "28.50"),
        new XElement("publish_date", "2004-12-22"),
        new XElement("description", "Jack is hunting for bad guys.")
    ),
    new XElement("book",
        new XAttribute("id", "Moonwalker01"),
        new XElement("author", "Michael Jackson"),
        new XElement("title", "24 Hours on the Moon"),
        new XElement("genre", "Thriller"),
        new XElement("price", "59.90"),
        new XElement("publish_date", "2000-02-08"),
        new XElement("description", "Micheal is taking a walk on the moon.")
    )
);
books.Save(@"..\..\..\newbooks.xml");
```

## XML-Daten einfügen

```
XDocument books = XDocument.Load(@"..\..\..\books.xml");

books.Element("catalog").Add(
    new XElement("book",
        new XAttribute("id", "Moonwalker"),
        new XElement("author", "Michael Jackson"),
        new XElement("title", "48 Hours on the Moon"),
        new XElement("genre", "Thriller"),
        new XElement("price", "59.90"),
        new XElement("publish_date", "2000-02-08"),
        new XElement("description", "Micheal is taking a walk on the moon.")
    )
);
books.Save(@"..\..\..\addbooks.xml");
```

### Datei addbooks.xml:

```
environment.</description>
</book>
<book id="Moonwalker">
  <author>Michael Jackson</author>
  <title>48 Hours on the Moon</title>
  <genre>Thriller</genre>
  <price>59.90</price>
  <publish_date>2000-02-08</publish_date>
  <description>Micheal is taking a walk on the moon.</description>
</book>
</catalog>
```

## XML-Daten modifizieren

Datei modifiedBooks.xml:

```
XDocument books = XDocument.Load(@"..\..\..\books.xml");

XElement root = books.Root;
root.Elements("book").Where(e => e.Attribute("id").Value == "bk103")
    .Select(e => e.Element("price")).Single().SetValue("9.95");

root.Elements("book").Where(e => e.Attribute("id").Value == "bk104")
    .Select(e => e.Attribute("id")).Single().Value = "myBK999";

books.Save(@"..\..\..\modifiedBooks.xml");
```

```
<book id="bk103">
  <author>Corets, Eva</author>
  <title>Maeve Ascendant</title>
  <genre>Fantasy</genre>
  <price>9.95</price>
  <publish_date>2000-11-17</publish_date>
  <description>After the collapse of a nanotechnology
    society in England, the young survivors lay the
    foundation for a new society.</description>
</book>
<book id="myBK999">
  <author>Corets, Eva</author>
  <title>Oberon's Legacy</title>
  <genre>Fantasy</genre>
  <price>5.95</price>
  <publish_date>2001-03-10</publish_date>
  <description>In post-apocalypse England, the mysterious
    agent known only as Oberon helps to create a new life
    for the inhabitants of London. Sequel to Maeve
    Ascendant.</description>
</book>
```

```
XDocument books = XDocument.Load(@"..\..\..\books.xml");

XElement root = books.Root;
root.Elements("book").Where(e => e.Attribute("id").Value == "bk103")
    .Select(e => e).Single().Remove();

books.Save(@"..\..\..\removeBooks.xml");
```

## XML-Daten löschen

Datei removeBooks.xml:

```
</book>
<book id="bk102">
  <author>Ralls, Kim</author>
  <title>Midnight Rain</title>
  <genre>Fantasy</genre>
  <price>15.95</price>
  <publish_date>2000-12-16</publish_date>
  <description>A former architect battles corporate zombies,
    an evil sorceress, and her own childhood to become queen
    of the world.</description>
</book>
<book id="bk104">
  <author>Corets, Eva</author>
  <title>Oberon's Legacy</title>
  <genre>Fantasy</genre>
  <price>5.95</price>
  <publish_date>2001-03-10</publish_date>
  <description>In post-apocalypse England, the mysterious
    agent known only as Oberon helps to create a new life
    for the inhabitants of London. Sequel to Maeve
    Ascendant.</description>
</book>
```

# Übung 1.3



## Bücher verarbeiten

- Verwende die XML Beispieldatei books.xml  
Link: <http://msdn.microsoft.com/en-us/library/ms762271%28VS.85%29.aspx>
- A) Schreibe eine «Windows Form»-Applikation, um die XML-Datei zu lesen und darzustellen
- B) Erweitere die Applikation, um die Daten als CSV zu speichern. Prüfe, ob die Datei mit Excel importiert werden kann.
- C) Schreibe die Daten als XML-Datei.
- D) Schreibe nur die Bücher mit Preis  $< 20.00$  in die XML-Datei.

# Übung 1.3



## Bücher verarbeiten

- E) Ergänze zu jedem Book ein neues Attribut ISBN
- F) Ergänze zu jedem Book ein neues Element `<publishing_location>` mit Angabe einer Stadt.

# XML

## XML-Dateien lesen mit XmlTextReader

### Beispielcode:

```
XmlTextReader reader = new XmlTextReader(@"..\..\books.xml");
while (reader.Read())
{
    if (reader.NodeType == XmlNodeType.Element)
    {
        if (reader.Name == "catalog")
        {
            Console.WriteLine("CATALOG");
        }
        if (reader.Name == "book")
        {
            if (reader.MoveToAttribute("id") == true)
            {
                Console.WriteLine("Book: Id = {0}", reader.Value.ToString());
            }
        }
        if (reader.Name == "author")
        {
            Console.WriteLine("  Author : {0}", reader.ReadElementContentAsString());
        }
    }
}
```

XML-Datei: books.xml

```
<?xml version="1.0"?>
<catalog>
  <book id="bk101">
    <author>Gambardella, Matthew</author>
    <title>XML Developer's Guide</title>
    <genre>Computer</genre>
    <price>44.95</price>
    <publish_date>2000-10-01</publish_date>
    <description>An in-depth look at creating applications
    with XML.</description>
  </book>
  <book id="bk102">
    <author>Ralls, Kim</author>
    <title>Midnight Rain</title>
    <genre>Fantasy</genre>
    <price>15.95</price>
    <publish_date>2000-12-16</publish_date>
    <description>A former architect battles corporate zombies,
    an evil sorceress, and her own childhood to become queen
    of the world.</description>
```

### Ausgabe:

```
CATALOG
Book: Id = bk101
  Author : Gambardella, Matthew
Book: Id = bk102
  Author : Ralls, Kim
Book: Id = bk103
  Author : Corets, Eva
Book: Id = bk104
  Author : Corets, Eva
```

# XML

## XML-Dateien schreiben mit XmlTextWriter

### Codebeispiel

```
XmlTextWriter writer = new XmlTextWriter(@"..\..\out.xml", Encoding.UTF8 );  
  
writer.Formatting = Formatting.Indented; ←  
writer.WriteStartElement("catalog"); ←  
writer.WriteStartElement("book");  
writer.WriteAttributeString("id", "myBookId"); ←  
writer.WriteStartElement("author");  
writer.WriteValue("Stephen King");  
writer.WriteEndElement();  
writer.WriteEndElement();  
writer.WriteEndElement();  
writer.Flush(); ←  
writer.Close();
```

Formatierung einschalten

Element schreiben

Attribut schreiben

Schreiben in Datei erzwingen

### XML-Ausgabedatei: out.xml

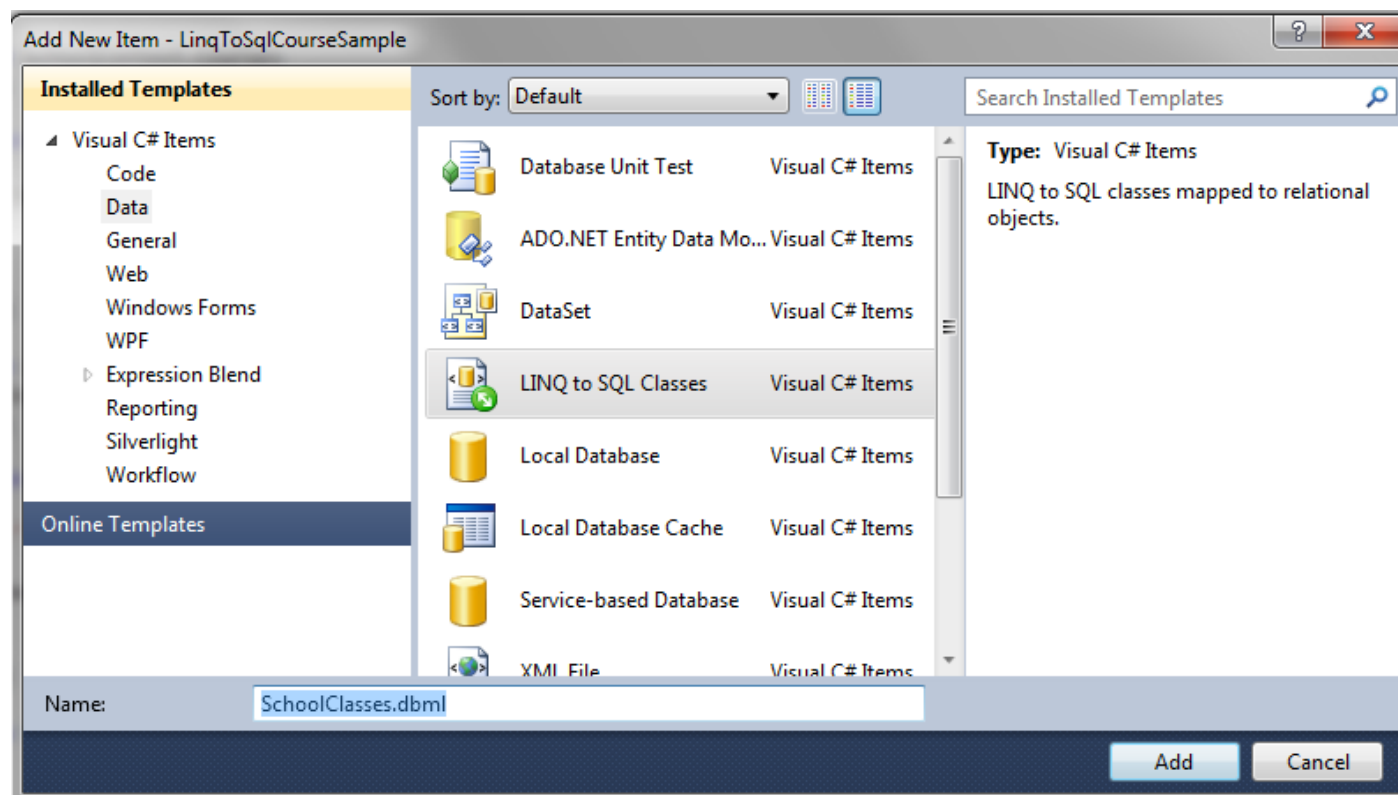
```
<catalog>  
  <book id="myBookId">  
    <author>Stephen King</author>  
  </book>  
</catalog>
```



## Einführung

- Ist ein O/R-Mapper-Framework, um mit LINQ sehr einfach direkt auf Daten einer Datenbank zugreifen zu können
- LINQ-to-SQL ist dem Entity Framework sehr ähnlich
- Wird von Microsoft nicht mehr weiter entwickelt

## LINQ-to-SQL-Klassen hinzufügen



## Anwendungsbeispiel: Konfigurationsdatei app.config

```
<?xml version="1.0"?>
<configuration>
  <appSettings>
    <add key="ConnString" value="Provider=Microsoft.Jet.OLEDB.4.0;
      Data Source=C:\Users\sab\Documents\60 HFU\2010\10 Exercises\13 ADO.NET\PersonenVerwaltung.mdb;
      Persist Security Info=False"/>
  </appSettings>
</configuration>
```

## Code zum Lesen des Connection-Strings

```
private string GetConnString()
{
    // Zudem braucht es eine Referenz auf System.Configuration
    return ConfigurationManager.AppSettings["ConnString"];
}
```

## Dem Modell die Datenbanktabellen hinzufügen

The screenshot shows the Microsoft Visual Studio (Administrator) interface for a project named 'LinqToSqlCourseSample'. The main design surface displays a LINQ-to-SQL model with four tables: Department, Course, OnlineCourse, and CourseInstructor. The Department table has properties: DepartmentID (primary key), Name, Budget, StartDate, and Administrator. The Course table has properties: CourseID (primary key), Title, Credits, and DepartmentID (foreign key). The OnlineCourse table has properties: CourseID (foreign key) and URL. The CourseInstructor table has properties: CourseID (foreign key) and PersonID (foreign key). The relationships are indicated by dashed lines with crow's foot notation. The Server Explorer on the left shows the database 'zrh0749\sqlexpress.School.d' with tables Course, CourseInstructor, Department, OfficeAssignment, OnlineCourse, and OnsiteCourse. The Solution Explorer on the right shows the project files, including SchoolClasses.dbml. The Properties window at the bottom right shows the properties for 'SchoolClassesDataContext', including Base Class (System.Data.Linq.DataContext), Connection (SchoolConnectionString), Context Namespace, Entity Namespace, Inheritance Modifier (None), and Name (SchoolClassesDataCont).

## Beispielcode (1/3) - Read

```
class Program
{
    static SchoolClassesDataContext schoolContext;

    static void Main(string[] args)
    {
        schoolContext = new SchoolClassesDataContext();
        Call Methods
    }

    static void ReadSample()
    {
        var result = from c in schoolContext.Courses
                     select c;

        foreach (Course item in result)
        {
            Console.WriteLine(item.CourseID + " " +
                              item.Title + " " + item.Department.Name);
        }
    }
}
```

## Beispielcode (2/3) – Update / Insert

```
static void UpdateSample()
{
    var result = from c in schoolContext.Courses
                  where c.CourseID == 1061
                  select c;

    foreach (Course item in result)
    {
        item.Title = "Upd-" + item.Title;
    }
    schoolContext.SubmitChanges();
}

static void InsertSample()
{
    Course newCourse = new Course()
    {
        CourseID = 9999,
        Title = "New Course",
        DepartmentID = 1,
        Credits = 8
    };

    schoolContext.Courses.InsertOnSubmit(newCourse);
    schoolContext.SubmitChanges();
}
```

## Beispielcode (3/3) - Delete

```
static void DeleteSample()
{
    var course = schoolContext.Courses.Where(c => c.CourseID == 9999).First<Course>();
    schoolContext.Courses.DeleteOnSubmit(course);
    schoolContext.SubmitChanges();
}
```

# Nützliche Links

---

## LINQ-Beispiele

- <https://docs.microsoft.com/en-us/samples/dotnet/try-samples/101-linq-samples/>
- <https://github.com/dotnet/try-samples/tree/master/101-linq-samples>

## PLINQ

- <https://docs.microsoft.com/en-us/dotnet/standard/parallel-programming/parallel-linq-plinq>