

LINQ

LANGUAGE **I**NTEGREATED **Q**UERY
.net 3.5

Bernhard Grojer
BernhardG@ppedv.de

Agenda

- Übersicht LINQ
- Basistechnologien
- Verschiedene Arten von LINQ
 - LINQ (to Objects)
 - LINQ TO SQL

Übersicht LINQ

- Aggregationen
- Typ Checking
- Einheitliche Filter-/Sortiermöglichkeiten
- Einheitliche Abfragesprache
 - XML
 - SQL
 - ...

C#

VB

Others...

.Net Language Integrated Query (LINQ)

LINQ enabled data sources

LINQ enabled ADO.NET

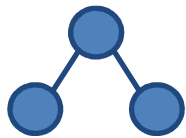
LINQ
To Objects

LINQ
To Datasets

LINQ
To SQL

LINQ
To Entities

LINQ
To XML



Objects



Relational

```
<book>  
<title/>  
<author/>  
<price/>  
</book>
```

XML

Basistechnologien - LINQ

COMPILER: C# 3.0 oder VB 9.0, ...

- *IEnumerable<T>**
- var Keyword
- Anonymous Types
- Lambda Expression
- Extension Methods
- Objekt-Initialisierung

* Gab es bereits in .net 2.0

Basistechnologien – LINQ

IEnumerable<T>

- Collection mit Interface `IEnumerable<T>`
 - Ist Voraussetzung für LINQ Abfragen
- LINQ Querys retournieren `IEnumerable<T>`

Namespace `System.Collection.Generic` bietet viele varianten von Collections die `IEnumerable<T>` implementieren!

Basistechnologien – LINQ

Objekt-Initialisierung

- Einfachere Objektinitialisierung OHNE Konstruktor

```
class Person {  
    public string FirstName;  
    public string LastName;  
    public int Age;  
}  
Person p = new Person{FirstName=„Max“, Age=10}
```

Basistechnologien – LINQ

var Keyword

```
var btn = new Button();
```

- Kein VARIANT Variablentyp wie in Java/VB 6
- Typisiert (erste Zuweisung legt Datentyp fest)
- IntelliSense funktioniert!

```
var i = 21; //Datentyp integer
```

```
i = „Test“; //Exception weil Typ bereits int ist!
```

Basistechnologien – LINQ

Anonymous Types

```
var per = new { FirstName=„Max“,  
               LastName=„Mustermann“ ,  
               Age = 123 };  
Console.WriteLine(per.FirstName);
```

Keine
Typangabe


- Klasse wird automatisch erstellt mit erforderlichen Properties (Internal Class)
- Erstellung mit Angabe des var Keyword
- Datentypen der Properties wird automatisch festgelegt anhand Zuweisung
- Interessant für „Reporting“ und Abfragen
- IntelliSense funktioniert auch hier

Basistechnologien – LINQ

Extension Methods

- Statische Methoden
- Erweitern existierender Typen (auch abstract!)
- Deklaration mit Keyword „**this**“

```
public static class Extensions
{
    public static string Reverse(this string s)
    {
        string strRes = string.Empty;
        for(int i = s.Length - 1; i >= 0; i--)
        {
            strRes += s[i];
        }
        return s;
    }
}
...
string strTest = „Hallo“;
Console.WriteLine(strTest.Reverse());
```



Basistechnologien – LINQ

Lambda Expression

- *Bisher möglich:*

- *Anonyme Methoden*

```
Func<Person, bool> MyFunc = delegate(Person p) { return p.FirstName.Length > 3; }
```

- *Benannte Methoden*

```
public bool MyFunction(string s)  
{  
    return true;  
}
```

```
Func<Person, bool> MyFunc = MyFunction;
```

```
List<Person> res = persons.FindAll(MyFunc);
```



Basistechnologien – LINQ

Lambda Expression

- Code Fragmente – mit Delegates Vergleichbar
Code wird als Parameter übergeben und ausgeführt

```
var res = persons.FindAll(p => p.FirstName.Length > 3);
```

Typ muss nicht nochmals
angeben werden

Expression
Context

Beispiel(e):

```
MyResult int = MyList.Sum(d => d.Quantity * d.Count);
```

```
List<Car> res = MyList.FindAll(c => c.MaxSpeed > 100);
```

```
MyList.ForEach(c => Console.WriteLine(c.MaxSpeed));
```

LINQ (TO OBJECT)

- SQL ähnliche Abfragesprache
- Strongly Typed
- Standard-Operatoren: from, where, orderby, select, ...


```
var res = from c in Customers  
where c.FirstName = „Max“  
orderby c.LastName  
select c;
```

IEnumerable<T>

Queryoperators

Sequenz (List)
return Type

LINQ (TO OBJECT)

```
IEnumerable<string> res =  
    from c in Customers  
    where c.FirstName = „Max“  
    select c.LastName.ToUpper();  
  
for (var s in res)  AUSFÜHRUNG  
    Console.WriteLine(s);
```

- Rückgabe kann modifiziert werden und beliebige Daten zurückliefern
- Ausführung erst bei Zugriff! (Verhalten kann mit ToList(), ToArray(), ... Methoden umgangen werden)

LINQ (TO OBJECT)

- LINQ kombiniert die Technologien (Lambda Expression, var Keyword, Anonymous Types, ...)

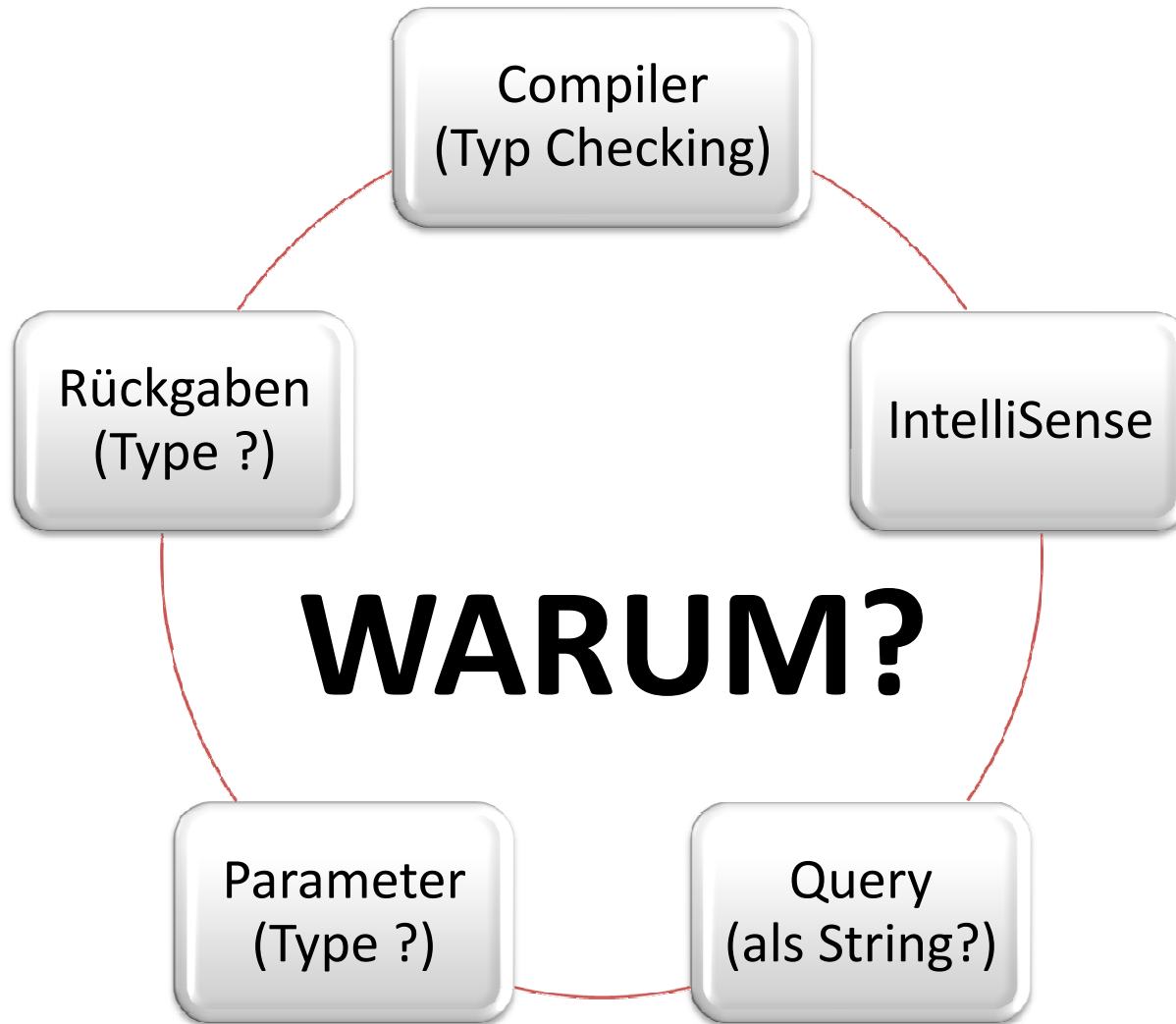
```
var res = from c in Customers
           where c.Orders.Sum(o => o.Quantity * o.Price) > 100
           orderby c.LastName, c.FirstName
           select new { Name = c.CompanyName,
                       Total = c.Orders.Count() };
```

```
res.ForEach(r => Console.WriteLine(r.CompanyName));
```

LINQ TO SQL

Problem:
Daten != Objekte

LINQ TO SQL



LINQ TO SQL

- DataAccess-API
- O/R Mapping als Aufsatz auf LINQ
 - LINQ TO SQL FILE
 - Codegeneration
 - Manuell via Attribute
 - SQLMETAL.EXE
 - Codegeneration

LINQ TO SQL

DataContext & Mapping

- DataContext
 - Schnittstelle zur Datenbank
 - Log
 - Transaktionen
 - Connection
 - Mapping
 - Patterns: Unit Of Work, Identity Map, Lazy Load, ...
 - ...

LINQ TO SQL

DataContext & Mapping

- Mapping (Relational <-> Objects)
 - Von ...
 - Tabellen
 - Feldern
 - Beziehungen
 - Über ...
 - Attribute
 - Bei der Klasse
 - XML
 - MappingSource im DataContext

LINQ TO SQL

Codegeneration

- Codegeneration
 - Drag'n'Drop und festlegen Property's, Relations (LINQ TO SQL)
 - Codegeneration anhand der Einstellungen
 - Automatisch von Tabellen (LINQ TO SQL)
 - Generieren von Tabellen
 - Code
 - Automatisch von Datenbanken
 - SQLMetal.exe
 - Mapping.xml
 - Code (.cs / .vb)

LINQ TO SQL

Insert, Delete, Update

- Insert
 - Add() bei Context
 - Collection
 - Table<T>
 - DB (Return) Value wird gemappt aufs Object
 - `DataContext.SubmitChanges();`
 - Context beinhaltet Eintrag erst nach `SubmitChanges();`

LINQ TO SQL

Insert, Delete, Update

- Update
 - Direkt beim Objekt
 - IdentityMap
 - ChangeTracking - `INotifyPropertyChanged` (optional)
 - Manuelles Verbinden des Context mit Entity
 - `DataContext.Customers.Attach(c);`
 - `DataContext.SubmitChanges();`

LINQ TO SQL

Insert, Delete, Update

- Delete
 - Remove bei Context
 - Collection
 - Table<T>
 - Remove => Entfernen des Entity oder der Beziehung
 - DataContext Cache
 - Beinhaltet im Cache das gelöschte Objekt selbst nach `DataContext.SubmitChanges();`
 - => Neu instanzieren des Context

LINQ TO SQL

Joins

- Joins
 - Alternativ zu DeferredLoadingEnabled (LazyLoad)
 - DataLoadOptions
 - Abbilden von Joins
 - CrossJoins (mehrere Select Keywords)
 - „Join“ Keyword
- => Automatische Generierung des TSQL Codes

LINQ TO SQL

DBNULL

- DBNULL
 - Nullable<T>
 - „.net Like“ -> `e.Region == null`
 - Codegeneration verwendet `Nullable<T>` je nachDatenbankfeld konfiguration

LINQ TO SQL

Ausführen v. Abfragen

- Zeitpunkt der „Ausführung“
 - Erst bei Zugriff auf die Daten
- `ToList()`
 - Sofortiges ausführen.
- Abfragen dynamisch „erweitern“
 - Ausführung auch hier erst bei Zugriff auf die Daten

LINQ TO SQL

StoredProcedures, SQL Functions

– StoredProcedures

- LINQ TO SQL File
 - Drag'n'Drop StoredProcedures
 - => Mapping und return Entity wird generiert
- DataContext.StoredProcedure(...);

– SQL Functions

- T-SQL (LEN Strings, Count, Math, DateTime, ...)

LINQ TO SQL

Transactions

– Transactions

- Context verwendet immer Transaktionen bei `SubmitChanges()`
- `TransactionScope` für mehrere `SubmitChanges()`
- `Transaction` Property des `DataContext`

LINQ TO ...?

- Interface IQueryable<T>
- Expression Trees
 - Vergleichbar mit Lambda Expressions, allerdings nicht bei der „Zuweisung“ ausgeführt sondern beim ersten Zugriff.

Expression<Func<Person, bool>>

=> LINQ TO AMAZON, LINQ TO LDAP, ...

ADO.net vNext

ADO.net vNext (August 2006 CTP) kommt nicht mit .net 3.5 sondern wird später nachgeliefert.

- LINQ TO ENTITY
 - Erweiterte Data-Mapping Funktionalitäten.
 - Mapping über mehre Tabellen, Spalten hinweg
 - Zusätzliche Flexibilität: LazyLoad, Property renaming, ...

FAZIT

- Einheitliche Abfragesprache
- Strongly Typed / IntelliSense
- O/R Mapper Ersatz
- RAD Design Support ab Beta 2 (LINQ Datasource)
- Optimierung möglich