Handguide
C# und Visual Studio

Autor: Dominik Hug

Inhalt

1 Grundlagen

1.1 Der new Operator

```
Article article = new Article("Computer");
```

Bei der Ausführung dieses Codes bewirkt der new Operator folgendes:

1. Er berechnet, wieviele Bytes für Instanzfelder (inklusive jenen der Obertypen) benötigt werden.
2. Er reserviert auf dem Heap entsprechenden Arbeitsspeicher und initialisiert alle Bytes mit 0.
3. Er ruft den Instanzenkonstruktor des Typs auf und übergibt ihm die Parameter. Jeder Konstruktor ist dafür verantwortlich, die Instanzfelder des Typs zu initialisieren und zuvor den Konstruktor der Oberklasse aufzurufen.

1.2 Strings

1.2.1 String.Empty

Die Klasse String hat die statische Eigenschaft `Empty`, diese entspricht "". `String.Empty == ""` evaluiert sich also zu `true`.

1.2.2 String.Format()

Mit der Methode `Format()` lassen sich numerische oder Datumswerte in einem String bequem formatieren.

Beispiel:

```
String.Format("Das Datum ist {0}, es sind {1:N} Ticks vergangen",
       DateTime.Now, DateTime.Now.Ticks)
```

Diese Methode hat den folgenden Rückgabewert:

```
Das Datum ist 17.07.2007 13:00:55, es sind 633'202'740'551'302'554.00 Ticks vergangen
```

Die Methode übernimmt als ersten Parameter den String, welcher formatierte Zahlen enthalten soll, danach eine beliebige Anzahl Objekte, die im String dargestellt werden sollen. In diesem String kann man die Position der zu formatierenden Objekte mit geschweiften Klammern die mit einer Nummer versehen sind platzieren.
Optional kann man in den geschweiften Klammern hinter einem Doppelpunkt ein Format angeben.
Formate:

Format:	anwendbar auf:	Beschreibung:
C	numerische Typen	Formatiert die Zahl als Währung (abhängig von der Kultur im CurrentThread)
E	numerische Typen	Wissenschaftliche Notation (z.B. `4.836E+003`)
F	numerische Typen	Zahl wird auf eine bestimmte Anzahl Dezimalstellen gerundet (z.B. {0:F3} bedeutet, dass der erste Parameter auf drei Dezimalstellen gerundet wird).
N	numerische Typen	Zahl wird kulturspezifisch dargestellt, abhängig vom CurrentThread.
P	numerische Typen	Darstellung als Prozentwert
X	Integer Typen	Darstellung als Hexadezimalwert
`dd.MM.yyyy` usw.	DateTime	zur Formatierung von Zeit- und Datumswerten, für genaue Beschreibung der Formatierungsmöglichkeiten siehe hier.

Übergibt man dieser Methode nicht formatierbare Objekte (d.h. Objekte die nicht `IFormattable` implementieren), ruft `String.Format()` deren `ToString()`-Methode auf.

1.3 Boxing und Unboxing

Werttypen werden auf dem Stack gespeichert, Referenztypen auf dem Heap. Klassen sind Referenztypen, während Strukturen Werttypen sind. Überall wo ein Referenztyp benötigt wird, kann man auch einen Werttyp verwenden (z.B. ein `int` kann einer `Object`-Variablen zugewiesen werden). Die automatische Konvertierung eines Werttyps in einen Referenztyp nennt man **Boxing**. Die Umkehroperation (Konvertierung eines Referenztyps in einen Werttyp) nennt man **Unboxing**.

Beispiele für Boxing und Unboxing:

```
object obj = 2; // Boxing

ArrayList list = new ArrayList();
list.Add(3); // Boxing
int i = (int)list[0] // Unboxing
```

1.4 Strukturierte Ausnahmebehandlung

```
public int kontonummer(string bezeichnung) {
   OleDbCommand cmdKonto = conn.CreateCommand();
   cmdKonto.CommandText = "SELECT Konto FROM tbl_Konto WHERE Bezeichnung = '" + bezeichnung + "'";
   int retVal;
   conn.Open();
   try {
      retVal = (int)cmdKonto.ExecuteNonQuery();
   }
   catch (Exception ex) {
      // kein passender Datensatz gefunden: 0 zurückgeben
      return 0;
   }
   finally {
      conn.Close(); // wird ausgeführt obwohl im catch-Block ein return-Statement
   }
   return retVal; // wird nur ausgeführt, wenn keine Exception auftritt
}
```

Normalerweise gilt, dass ein `return`-Statement die letzte Anweisung ist die in einer Methode ausgeführt wird. Ein `finally`-Block bildet hier eine Ausnahme.

1.5 Syntax-Details

1.5.1 switch-Anweisung

Mehrere Werte in einem case-Block:

```
switch (myVariable) {
   case "Artikel":
   case "Mon":
   case "Trans":
      ...
   break;
}
```

2 Neues in C# 2

2.1 Anonyme Methoden

Wird eine Methode als Event-Handler für nur ein einziges Event verwendet, bieten sich anonyme Methoden an. Vorteil: Die Methode ist an der gleichen Stelle implementiert, an welcher sie mit dem Event verknüpft wird.

Beispiel:

```
// Voraussetzung: lblBetrag hat eine Datenbindung
lblBetrag.DataBindings[0].Format += delegate(object sender, ConvertEventArgs e) {
    double betrag = (double)e.Value;
    e.Value = betrag.ToString("N2");
};
```

Verwendet man in der anonymen Methode keine der vom Event übergebenen Argumente kann man die Parameterliste weglassen.

Beispiel:

```
button1.Click += delegate { MessageBox.Show("Sie haben geklickt"); };
```

2.2 Properties: Verschiedene Sichtbarkeit für Getter und Setter

In C# 1 war es nicht möglich, ein Property mit öffentlichem Getter aber privatem Setter zu schreiben (resp. andere Kombinationen aus (`public`, `private`, `protected`, `internal`, `protected internal`). Man konnte sich abhelfen, indem man ein readonly Property (nur Getter) implementierte und eine private `SetXXX()`-Methode schrieb. Das führte jedoch nicht gerade zu schönem Code.

C# 2 löst dieses Problem, indem man dem Getter oder Setter explizit eine eingeschränktere Sichtbarkeit geben kann.

Beispiel

```
private string _test;
public string Test {
    get { return _test; }
    private set { _test = value; }
}
```

Es ist jedoch nicht möglich, die Sichtbarkeit des Getters oder Setter grösser als die des Properties zu definieren.
Anmerkung: Dies ist die einzige Stelle an welcher das Schlüsselwort `private` wirklich benötigt wird, da die Standardsichtbarkeit für alle Elemente der eingeschränkteste aller möglichen Sichtbarkeitsmodifizierer ist.

2.3 Alias für den globalen Namespace

Situation: Die Klasse `User` existiert im globalen Namespace (nicht innerhalb eines `namespace { … }` Blocks) und innerhalb des Namespaces `ch.dominikhug`.
Innherhalb des Namespaces ch.dominikhug gab es bei C# 1 keine Möglichkeit die Klasse User im globalen Namespace anzusprechen. C# 2 löst dieses Problem mit dem Alias `global` für den globalen Namespace.
Beispiel:

```
using System;

class User { } // Klasse im globalen Namespace

namespace ch.dominikhug {

    class User { }

    class Program {
        static void Main(string[] args) {
            Console.WriteLine(typeof(User));
            Console.WriteLine(typeof(global::User)); // kann nur dank "global::" angesprochen werden
        }
    }
}
```

Die Ausgabe des Programms ist:

```
ch.dominikhug.User
User
```

2.4 Generische Typen

Vorteile:

- Der Compiler kann die Typsicherheit stärker prüfen
- Die IDE kann dank der Extra-Information Intelli-Sence Optionen anzeigen
- Aufrufer von Methoden können sicher sein, dass sie korrekte Argumente übergeben
- Der Just In Time Compiler kann mit Strukturen besser umgehen und d.h. Boxing und Unboxing vermeiden
- Der Code ist besser lesbar, die Absichten des Programmierers werden klarer ausgedrückt

2.4.1 List<T>

Die generische Klasse List<T> bietet eine typsichere Collection. Bei der Deklaration geben wir an, welchen Typ ihre Elemente haben. Es ist nicht möglich, Elemente eines anderen Typs hinzuzufügen.

Beispiel: Wir haben eine Collection mit Personennamen und wollen alle ausgeben, welche mehr als 5 Zeichen umfassen.
Ohne generische Collection:

```
ArrayList lstNamen = new ArrayList();
lstNamen.AddRange(new string[] { "Sarah", "Joseph", "Simon", "Gerhard" });
// hier könnte ein beliebiger Typ zu lstNamen hinzugefügt werden
// was weiter unten zu einem Laufzeitfehler führt
lstNamen.Add(4);
for (int i = 0; i < lstNamen.Count; i++) {
    string name = (string)lstNamen[i]; // hier ist ein Cast notwendig
    if (name.Length > 5) Console.WriteLine(name);
}
```

Mit List<T>:

```
// jedem Leser wird klar, dass lstNamen nur String-Objekte enthalten wird
List<string> lstNamen = new List<string>();
lstNamen.AddRange(new string[] { "Sarah", "Joseph", "Simon", "Gerhard" });
// Versucht man hier einen anderen Typ als string hinzuzufügen
// kommt es zu einem Compilerfehler
for (int i = 0; i < lstNamen.Count; i++) {
    // lstNamen[i] gibt direkt einen String zurück
    if (lstNamen[i].Length > 5) Console.WriteLine(lstNamen[i]);
}
```

Die `foreach` Anweisung ermöglicht zwar auch ohne generische Collection den expliziten Cast wegzulassen, dennoch kommt es zu einem impliziten Cast!

```
ArrayList lstNamen = new ArrayList(); // nochmals mit der "alten" ArrayList
lstNamen.AddRange(new string[] { "Sarah", "Joseph", "Simon", "Gerhard" });
// hier kommt es zu einem impliziten Cast, da ArrayList immer ein Object zurückgibt
foreach (string name in lstNamen) {
    if (name.Length > 5) Console.WriteLine(name);
}
```

2.4.2 Methoden von List<T>

`public void` **`RemoveAll`**`(Predicate<T> match)`

Entfernt alle Elemente aus der Liste, die einer bestimmten Bedingung genügen. `Predicate<T>` ist ein Delegat (also ein Zeiger auf eine Methode), welche einen bool'schen Wert zurückgibt. `FindAll()` übernimmt als Argument also eine Methode mit bool'schem Rückgabewert die ein Argument vom Typ T übernimmt. Am elegantesten ist die Übergabe einer anonymen Methode.

Beispiel

```
List<string> lstNamen = new List<string>();
lstNamen.AddRange(new string[] { "Sarah", "Joseph", "Simon", "Gerhard" });
// der Methode RemoveAll() eine anonyme Methode als Argument übergeben
lstNamen.RemoveAll(delegate(string name) { return name.Length > 5; });
foreach (string name in lstNamen) {
    Console.WriteLine(name);
}
```

Existiert eine bool'sche Methode mit einem String als Argument, können wir der Methdoe `RemoveAll()` auch die Methode direkt übergeben.

Beispiel

```
static bool Check(string name) {
    return name.Length > 5;
}
...
List<string> lstNamen = new List<string>();
lstNamen.AddRange(new string[] { "Sarah", "Joseph", "Simon", "Gerhard" });

lstNamen.RemoveAll(Check);
foreach (string name in lstNamen) {
    Console.WriteLine(name);
}
```

Dies geht auch mit nicht statischen Methoden. Dann müssen wir zuerst die entsprechende Klasse instanziieren und `RemoveAll(`*`instanzenname`*`.Check)` aufrufen.

`public bool` **`Exists`**`(Predicate<T> match)`

Gibt `true` zurück, fall ein Element existiert welches der Bedingung `match` genügt.

Beispiel

```
List<Person> lstPerson = new List<Person>();
lstPerson.Add(new Person("Heinz", "Schuhmacher"));
lstPerson.Add(new Person("Otto", "Meier"));
lstPerson.Add(new Person("Peter", "Zwyssig"));
lstPerson.Add(new Person("Friedrich", "Meier"));

Console.WriteLine(lstPerson.Exists(delegate(Person person) {
    return person.FirstName == "Heinz" && person.FamilyName == "Schuhmacher";
}));
```

`public void` **`Sort`**`(Comparison<T> comparison)`

Diese Überladung von `Sort()` ermöglicht uns die Sortierung der Collection mit einem eigenen Vergleichsalgorithmus. Diesen können wir mit einer anonymen Methode direkt im Methodenaufruf implementieren.

`Comparison<T>` ist ein Delegat, der auf eine Methode zeigt, welche zwei Instanzen von T übernimmt und einen `int` zurückgibt. Der Rückgabewert ist kleiner als 0 fallst das erste Argument in der Sortierreihenfolge kleiner als das zweite ist, 0 falls die Argumente gleich sind, grösser als 0 wenn das erste Argument grösser als das zweite ist.

Beispiel

Eine Liste von Personen soll sortiert werden, zuerst nach Familienname, dann nach Vorname. Die anonyme Methode ist fett formatiert.

```
List<Person> lstPerson = new List<Person>();
lstPerson.Add(new Person("Heinz", "Schuhmacher"));
```

```
lstPerson.Add(new Person("Otto", "Meier"));
lstPerson.Add(new Person("Peter", "Zwyssig"));
lstPerson.Add(new Person("Friedrich", "Meier"));

lstPerson.Sort(delegate(Person x, Person y) {
    // falls Nachname gleich, Vornamen vergleichen
    if (x.FamilyName == y.FamilyName) return x.FirstName.CompareTo(y.FirstName);
    // sonst: nur Nachnamen vergleichen
    else return x.FamilyName.CompareTo(y.FamilyName);
});

foreach (Person person in lstPerson) {
    Console.WriteLine(person.FamilyName + " " + person.FirstName);
}
```

Die Ausgabe ist:

```
Meier Friedrich
Meier Otto
Schuhmacher Heinz
Zwyssig Peter
```

2.5 yield return

Das Keyword `yield` vereinfacht das Implementieren von Enumeratoren. Wir wollen eine Methode `Zweierpotenzen(int exponent)` schreiben, welche einen Enumerator mit allen Zweierpotenzen bis zu einem bestimmten Exponenten zurückgibt (z.B. `Zweierpotenzen(3)` soll 2, 4, 8 zurückgeben).

Beispiel

```
class Program {

    static IEnumerable Zweierpotenzen(int exponent) {
        int result = 1;
        for (int counter = 1; counter <= exponent; counter++) {
            result *= 2;
            yield return result;
        }
    }

    static void Main(string[] args) {
        foreach (int i in Zweierpotenzen(10)) {
            Console.Write("{0} ", i);
        }
    }

}
```

Die Ausgabe ist

```
2 4 8 16 32 64 128 256 512 1024
```

Der Compiler sorgt dafür, dass `Zweierpotenzen()` ausgeführt wird und jedesmal wenn nach `yield return` pausiert (aber der Zustand von `Zweierpotenzen()` wird gespeichert). Benötigt die `foreach` Anweisung das nächste Element wird die Ausführung von `Zweierpotenzen()` fortgeführt.

2.6 Nullables

`System.Nullable<T>` ist eine generische Struktur welche auch fehlende Werte (`null`) repräsentieren kann. Als Typparameter `T` dürfen nur Werttypen verwendet werden (Referenztypen machen hier keinen Sinn, da ihnen bereits null zugewiesen werden kann).
Die wichtigsten Properties von Nullable sind `HasValue` und `Value`. Ihre Bedeutung ist offensichtlich. Ist `HasValue` false und man verwendet `Value`, wird eine `InvalidOperationException` ausgelöst.
Nullable hat zwei Konstruktoren. Der parameterlose Standardkonstruktor erzeugt eine Instanz ohne Wert (null). Der andere übernimmt als Argument den zu repräsentierenden Wert.

Beispiel

```
Nullable<int> x;
// die folgenden Zeilen haben die gleiche Bedeutung
x = 5;
x = new Nullable<int>(5);
```

Die Methode `GetValueOrDefault()` hat zwei Überladungen. Beide geben den Wert zurück, falls einer vorhanden ist. Die parameterlose Variante gibt den Standardwert des darunter liegenden Typs zurück, falls kein Wert vorhanden ist (0 für numerische Typen, false für bool). Die andere Überladung ermöglicht die Übergabe eines Wertes welche zurück gegeben wird falls kein Wert vorhanden ist.

2.6.1 Boxing und Unboxing bei Nullables

Eine Instanz von `Nullable<T>` wird in eine Nullreferenz geboxt (falls kein Wert vorhanden ist) oder in einen geboxten Wert vom Typ `T`. Unboxing ist sowohl zu `Nullable<T>` oder zu `T` möglich.

Beispiel

```
Nullable<int> nullable = 5;

// Boxing zu int
object boxed = nullable;
// Gibt System.Int32 aus
Console.WriteLine(boxed.GetType());

// Unboxing zu normalem int
int normal = (int)boxed;
// Unboxing zu Nullable<int>
nullable = (Nullable<int>)boxed;
```

2.6.2 Syntaktischer Zucker für Nullables

Statt `Nullable<int>` können wir auch `int?` schreiben. Dies funktioniert für alle Typen. Beides wird zum exakt gleichen IL-Code kompiliert. Diese Kurzform kann überall verwendet werden, auch in Methodensignaturen, `typeof` Ausdrücken, Casts usw.

C# 2 führt auch den **null coalescating operator** ?? ein. Es handelt sich um einen binären Operator welcher `first ?? second` in folgenden Schritten evaluiert:

1. Evaluiere `first`.
2. Ist `first` nicht `null`, ist dies der Rückgabewert des Ausdrucks.
3. Andernfalls: evaluiere `second` und dies wird der Rückgabewert des Ausdrucks.

3 Neues in C# 3

3.1 Automatisch Implementierte Properties

```
string _name;
public string Name {
    get { return _name; }
    set { _name = value; }
}
```

kann jetzt geschrieben werden als

```
public string Name { get; set; }
```

Der Compiler generiert eine private Variable, welche im Code nicht direkt referenziert werden kann und implementiert Getter und Setter mit simplem Code für Lesen und Schreiben.

Auch verschiedene Sichtbarkeiten für Getter und Setter sind möglich:

```
public string Name { get; private set; }
```

3.2 Implizit typisierte lokale Variablen

Statt

```
Dictionary<string, int> dict = new Dictionary<string, int>();
```

kann man jetzt auch schreiben:

```
var dict = new Dictionary<string, int>();
```

Dies verhindert, dass der Typ zwei Mal im Code vorkommt.
Implizit typisierte lokale Variablen können nur verwendet werden wenn:

- Die Variable lokal deklariert wurde.
- Die Variable als Teil der Deklaration initialisiert wird.
- Die Initialisierung nicht `null` ist.
- In der Anweisung nur eine Variable deklariert wird.

Implizit typisierte lokale Variablen können auch in `using`, `for` und `foreach` Anweisungen verwendet werden.

WICHTIG:
Implizit typisierte lokale Variablen dürfen nicht mit Variants von Visual Basic 6 verglichen werden. Der Typ der Variable wird bei der Deklaration festgelegt und bleibt.

3.3 Vereinfachte Initialisierung

Unsere Beispieltypen:

```
class Adresse {
    public string Strasse { get; set; }
    public string Ort { get; set; }
}

class Person {
    public string Name { get; set; }
    public int Alter { get; set; }
    public Adresse Adresse { get; set; }

    List<Person> lstFreunde = new List<Person>();
    public List<Person> Freunde {
        get { return lstFreunde; }
    }

    public Person() {
        Adresse = new Adresse();
    }
    public Person(string name) {
        Name = name;
        Adresse = new Adresse();
    }
}
```

statt

```
Person pers = new Person();
pers.Name = "Peter";
pers.Alter = 21;
```

kann man jetzt schreiben:

```
Person pers = new Person() { Name = "Peter", Alter = 21 };
```

dank des parametrisierten Konstruktors ist auch möglich:

```
Person pers = new Person("Peter") { Alter = 21 };
```

Es kommt noch besser: Das Property Adresse hat einen komplexen Typ, auch diesen können wir so initialisieren:

```
Person pers = new Person("Tom") { Adresse = { Strasse = "Bahnhofstrasse", Ort = "Zürich" } };
```

3.4 Collection Initializers

```
List<string> lstNamen = new List<string>();
lstNamen.Add("Peter");
lstNamen.Add("Karl");
lstNamen.Add("Herbert");
lstNamen.Add("Paul");
```

lässt sich jetzt schreiben als:

```
List<string> lstNamen = new List<string> { "Peter", "Karl", "Herbert", "Paul" };
```

Solche Initialisierungen waren vor C# 3 nur mit Arrays möglich.

Auch dies ist möglich:

```
List<Person> lstPersonen = new List<Person> {
    new Person { Name="Peter", Alter = 21 },
    new Person { Name="Karl", Alter = 43 }
};
```

Jetzt können wir auch mit vereinfachter Initialisierung die Liste der Freunde einer Person initialisieren.

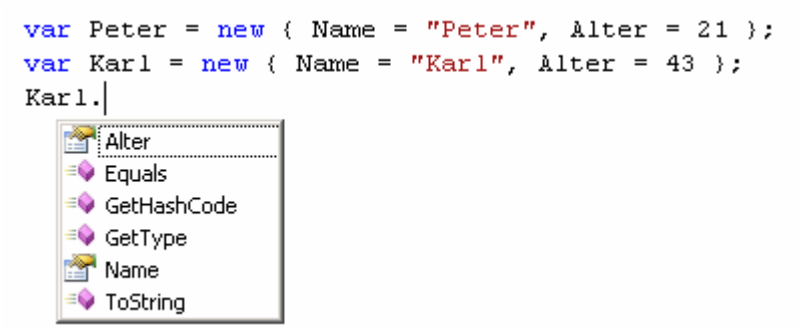
```
Person prs = new Person {
    Name = "Peter",
    Alter = 21,
    Freunde = {
        new Person { Name="Karl", Alter = 43 },
        new Person("Herbert") {
            Alter = 25,
            Adresse = { Strasse = "Bahnhofstrasse", Ort = "Zürich"}
        }
    }
};
```

3.5 Projection Initializers

```
List<Person> family = new List<Person>() {
    new Person() { Name = "Karl", Alter = 21 },
    new Person() { Name = "Susi", Alter = 34 },
    new Person() { Name = "Stefan", Alter = 12}
};

var converted = family.ConvertAll(delegate(Person person) {
    // der Name des ersten Properties dieses anonymen Typs muss nicht angegeben werden,
```

```
    // es wird automatisch Name genommen
    return new { person.Name, IstErwachsen = (person.Alter >= 18) };
});

foreach (var person in converted) {
    Console.WriteLine("Name = {0}, Erwachsen = {1}", person.Name, person.IstErwachsen);
}
```

3.6 Anonyme Typen



Mit den ersten beiden Zeilen haben wir einen anonymen Typ kreiert. Dieser hat keinen Namen, wohl aber die zwei Properties Name und Alter.

Dank implizit typisierten Arrays ist auch folgendes möglich:

```
var family = new[] {
    new { Name = "Karl", Alter = 21 },
    new { Name = "Susi", Alter = 34 },
    new { Name = "Stefan", Alter = 12}
};
int alterTotal = 0;
// Iterationsvariable muss implizit typisiert sein, da der Iterationstyp keinen Namen hat
foreach (var person in family) {
    alterTotal += person.Alter;
}
```

Die Properties der anonymen Typen sind readonly!

3.7 Extension Methods

Szenario:
Wir würden gerne der Klasse `String` eine neue Methode hinzufügen, welche zählt wie oft ein anderer String darin vorkommt. Dies ist aber nicht möglich, da String versiegelt (`sealed`) ist.

```
static class Extension {
    public static int Count(this string str, string toCount) {
        int count = 0;
        int temp = str.IndexOf(toCount);
        while (temp != -1) {
            count++;
            str = str.Substring(temp + 1);
            temp = str.IndexOf(toCount);
        }
        return count;
    }
}
…
string myString = "ABCABCABC";
Console.WriteLine(myString.Count("ABC"));
```

Mit Extension Methods wird dies möglich. Dazu definieren wir in einer statischen Klasse eine statische Methode mit gewünschtem Rückgabetyp (in diesem Fall int) und als erstem Parameter einen `string`. Diesen ersten Parameter versehen wir mit dem Schlüsselwort `this`.
Die Extension Method `Count()` können wir nun verwenden als wäre sie ein Member von `String`.

3.8 Lambda Expressions

Lambda Ausdrücke können als eine Evolution von anonymen Methoden angesehen werden. Es gibt fast nichts das mit anonymen Methoden erreicht werden kann, was nicht auch mit Lambda Ausdrücken gemacht werden kann. In fast jedem Fall sind die Lambdas kompakter und besser lesbar.

3.8.1 Func<…> Delegate Typen

Im System Namespace von .NET 3.5 gibt es fünf generische Delegate Typen. Hier sind ihre Signaturen:

```
public delegate TResult Func<TResult>()
public delegate TResult Func<T, TResult>(T arg)
public delegate TResult Func<T1, T2, TResult>(T1 arg1, T2 arg2)
public delegate TResult Func<T1, T2, T3, TResult>(T1 arg1, T2 arg2, T3 arg3)
public delegate TResult Func<T1, T2, T3, T4, TResult>(T1 arg1, T2 arg2, T3 arg3,
T4 arg4)
```

Der letzte Typparameter spezifiziert den Rückgabetyp des Delegates.
Beispiel:
`Func<string, double, int>` ist äquivalent zu
`delegate int SomeDelegate(string arg1, double arg2)`.

3.8.2 Transformation zu einem Lambda Ausdruck

```
Func<string, int> returnLength;
returnLength = delegate(string text) { return text.Length; };
```

In obigem Code ist die anonyme Methode fett hervorgehoben. Diese konvertieren wir jetzt in einen Lambda Ausdruck.

Die längste Form eines Lambdas hat folgende Syntax:
`(typisierte_Parameterliste) => { Ausdrücke }`
=> ist neu in C# 3 und teilt dem Compiler mit, dass wir einen Lambda Ausdruck benutzen.

```
Func<string, int> returnLength;
returnLength = (string text) => { return text.Length; };
```

Wiederum ist der Ausdruck zur Erzeugung des Delegates fett hervorgehoben.

Besteht der Rumpf der anonymen Methode lediglich aus einem einzigen Ausdruck, kann man die geschweiften Klammern und das `return` Keyword weglassen:

```
returnLength = (string text) => text.Length;
```

Eine **implizit typisierte Parameterliste** ist eine komma-separierte Liste von Namen. Es ist nicht möglich, in einer Parameterliste typisierte und implizit typisierte Parameter zu mischen. Falls die Liste `out` oder `ref` Parameter enthält, muss man die typisierte Version verwenden. Wir können unseren Ausdruck also weiter verkürzen zu:

```
returnLength = (text) => text.Length;
```

Benötigt der Lambda Ausdruck nur einen einzelnen Parameter, kann man die runden Klammern für die Parameterliste weglassen:

```
returnLength = text => text.Length;
```

Es ist möglich, den gesamten Lambda Ausdruck in runde Klammern zu setzen, was die Lesbarkeit leicht erhöht:

```
returnLength = (text => text.Length);
```

4 Formularprogrammierung

4.1 System.Windows.Forms.Control

Da Control die Oberklasse aller Steuerelemente ist, lohnt es sich dessen Member genau anzuschauen.

4.1.1 Wichtige Methoden von Control

`public Point` **`PointToClient`**`(Point p)`
Rechnet die Koordinaten eines Pixels von Bildschirm-Koordinaten in Client-Koordinaten (dem Koordinatensystem des Controls) um.

4.1.2 Wichtige Ereignisse von Control

`public event MouseEventHandler` **`MouseDown`**

Die in den `MouseEventArgs` übergebenen Mauskoordinaten sind die Client-Koordinaten (nicht das Koordinatensystem des Bildschirms, sondern das des Controls. Drückt man die Maus in der oberen linken Ecke des Controls, ist `e.X` und `e.Y` 0.
AUFGEPASST! Beim Ereignis `DragDrop` beziehen sich die Mauskoordinaten von `DragEventArgs` hingegen die Bildschirmkoordinaten. Um sie in Client-Koordinaten umzurechnen muss man die Methode `PointToClient()` aufrufen.

4.2 Form

Die Eigenschaft `AcceptButton` eines Formulars kann auf einen Button gesetzt werden. Dieser reagiert dann auf die RETURN-Taste, allerdings nur wenn man die Eigenschaft `DialogResult` dieses Buttons auf einen Wert setzt.
Buttons haben die Eigenschaft `DialogResult` welche auf Werte wie `OK` oder `Cancel` gesetzte werden kann. Zeigt man das Formular mit `ShowDialog()` an, ist es ein modaler Dialog. Die Methode `ShowDialog()` gibt dem Aufrufer den Wert des Buttons zurück, mit welchem das Formular geschlossen wurde. Setzt man bei einem Formular die Eigenschaft `DialogResult` eines Buttons, so wird das Formular bei Betätigung des Buttons automatisch geschlossen (ein Aufruf von `Close()` im Eventhandler des Buttons ist nicht notwendig).

4.2.1 Beim Anzeigen des Formulars Cursor setzen

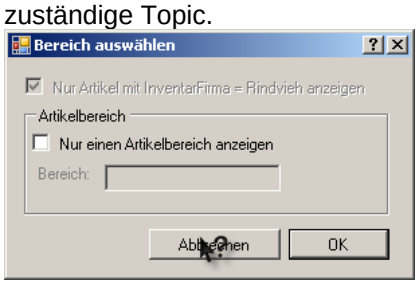
Im Event Shown die Methode `SetFocus()` des betreffenden Controls aufrufen.

4.2.2 Schliessen des Formulars

Das Event `FormClosing` wird vor dem Schliessen des Formulars ausgeführt. Im Eventhandler kann man das Schliessen verhindern indem man `e.Cancel` auf `true` setzt. Es wird dadurch auch das Schliessen des Formulars im Code durch `Close()` verhindert!

4.2.3 HelpButton Eigenschaft

Diese bool'sche Eigenschaft bestimmt, ob in der Titelleiste ein Hilfeknopf angezeigt wird (Box mit einem Fragezeichen). Der Standardwert ist `false`. Allerdings müssen die Eigenschaften `MaximizeBox` und `MinimizeBox` auf `false` gesetzt werden, damit der HelpButton sichtbar wird.
Klickt der User auf den Help-Button, erscheint rechts des Mauspfeiles ein Fragezeichen. Klickt der User auf ein Control, welches mit einem Hilfethema assoziiert ist so öffnet sich die **CHM-Datei** und zeigt das zuständige Topic.



Mit dem Formular-Event `HelpButtonClicked` kann man auch mit eigenem Code auf das Drücken des HelpButtons reagieren. Dabei kann man den "Hilfemodus" (das Fragezeichen rechts des Mauspfeils) durch `e.Cancel = true` wieder verlassen.
Der Vorteil dieser Methode ist, dass der User im "Hilfemodus" ein Steuerelement auswählen kann, das sonst nicht den Fokus erhalten kann, z.B. einen Button.
Dies funktioniert auch, wenn die Controls in einem Custom Control integriert sind, welches seinen eigenen HelpProvider hat.
Siehe auch: Integration von .CHM-Dateien in eine .NET Applikation

4.3 TextBox

4.3.1 Nur numerische Eingaben erlauben

Folgendes Beispiel erlaubt nur die Eingabe der Zeichen 0 – 9 in `textBox1`. Die Pfeiltasten und DELETE werden auch akzeptiert, nicht aber BACKSPACE.

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e) {
    if (!char.IsNumber(e.KeyChar)) {
        e.Handled = true; // bewirkt, dass der Tastendruck nicht weiter verarbeitet wird
```

```
    }
}
```

Der Code in diesem Beispiel ermöglicht dem User auch das Löschen mit BACKSPACE.

```
private void textBox1_KeyPress(object sender, KeyPressEventArgs e) {
    if (!char.IsNumber(e.KeyChar) && (Convert.ToInt32(e.KeyChar) != 8)) {
        e.Handled = true;
    }
}
```

Mit `Convert.ToInt32(e.KeyChar)` erhalten wir den ASCII-Code der gedrückten Taste. Der ASCII-Code von BACKSPACE ist 8.

4.4 CheckBox

Setzt man die `AutoCheck` Eigenschaft auf `false`, so kann der User den Zustand (Checked oder Unchecked) nicht mehr ändern. Diese Möglichkeit hat dann nur noch die Applikation resp. die Datenquelle an welche die CheckBox gebunden ist.

4.5 ComboBox

Setzt man die Eigenschaft `DropDownStyle` auf `DropDownList` (Standardeinstellung ist `DropDown`), ist es nicht möglich mit der Tastatur einen Text einzugeben.

4.5.1 SelectionChangeCommited-Ereignis

Das Ereignis `SelectionChangeCommited` wird nur ausgelöst, wenn der User selbst in der ComboBox einen anderen Eintrag auswählt (nicht jedoch wenn sich `SelectedValue` resp. `SelectedIndex` aufgrund der Datensatz-Navigation ändern). Somit entspricht `SelectionChangeCommited` dem Ereignis "Nach Aktualisierung" von Kombinationsfeldern in Access.

4.5.2 Datenbindung

ComboBoxes werden meist wie folgt gebunden.
Komplexe Datenbindung an eine Tabelle, deren Zeilen die Items der ComboBox darstellen (z.B. Verpackungsarten). Dazu setzt man die Eigenschaft `DataSource` im Eigenschaftenfenster auf das DataSet, und die Eigenschaften `DisplayMember` und `ValueMember` auf die entsprechenden Felder der Tabelle.
Einfache Datenbindung an eine Spalte der "Haupttabelle", z.B. Produkte mit einem Feld VerpackungsartID. Dazu geht man im Eigenschaftenfenster auf (DataBindings) und setzt `SelectedValue` auf das entsprechende Feld.
WICHTIG: in diesem Fall sollte man die Tabelle für die komplexe Datenbindung (im Beispiel Verpackungsarten) zuerst binden, erst danach die Tabelle mit der einfachen Datenbindung (Produkte), ansonsten wird die Verpackungsart nicht korrekt angezeigt.

4.6 TreeView

4.6.1 Drag & Drop innerhalb einer TreeView

Beispiel:

```
// e.X und e.Y sind die Client-Koordinaten
private void treeView_MouseDown(object sender, MouseEventArgs e) {
    TreeNode dragNode = treeView.GetNodeAt(new Point(e.X, e.Y));
    if (dragNode != null && dragNode == treeView.SelectedNode) {
        // zweiter Parameter legt fest, welche DragDropEffekte das Quellsteuerelement zulässt
        treeView.DoDragDrop(dragNode, DragDropEffects.All);
    }
}

// damit während des Ziehens das richtige Symbol erscheint
private void treeView_DragOver(object sender, DragEventArgs e) {
    if (e.KeyState == 9) { // linke Maustaste und Ctrl gedrückt
        e.Effect = DragDropEffects.Copy;
    }
    else {
        e.Effect = DragDropEffects.Move;
    }
}

// e.X und e.Y sind Screen-Koordinaten
private void treeView_DragDrop(object sender, DragEventArgs e) {
    TreeNode dropNode; // diesem Knoten wird der Drag-Drop-Knoten hinzugefügt
    Point clientPoint = treeView.PointToClient(new Point(e.X, e.Y));
```

```
    dropNode = treeView.GetNodeAt(clientPoint);
    if (dropNode != null) {
        TreeNode dragNode = (TreeNode)e.Data.GetData(typeof(TreeNode));

        // falls Zielknoten ein Unterknoten von Quellknoten ist abbrechen
        if (contains(dragNode, dropNode)) {
            return;
        }

        if (e.Effect == DragDropEffects.Copy) { // Kopieren
            // der Drag-Drop-Knoten muss geklont werden, da er nicht zwei Oberknoten haben kann
            TreeNode newNode = (TreeNode)dragNode.Clone();
            dropNode.Nodes.Add(newNode);
        }
        else { // Verschieben
            if (dragNode.Parent != null) {
                // Falls Vaterknoten vorhanden: Knoten vom Vaterknoten entfernen
                dragNode.Parent.Nodes.Remove(dragNode);
            }
            else {
                // Falls es sich um einen Wurzelknoten handelt: Knoten von TreeView entfernen
                treeView.Nodes.Remove(dragNode);
            }
            dropNode.Nodes.Add(dragNode); // dem neuen Vaterknoten hinzufügen
        }
        dropNode.ExpandAll();
    }
}

/// <summary>Gibt true zurück, falls node ein Kind, Enkel oder Urenkel... von containerNode
ist.</summary>
private bool contains(TreeNode containerNode, TreeNode node) {
    bool contains = false;
    TreeNode ancestorNode = node.Parent;
    while (ancestorNode != null) {
        if (ancestorNode == containerNode) {
            contains = true;
            break;
        }
        else {
            ancestorNode = ancestorNode.Parent;
        }
    }
    return contains;
}
```

4.7 ToolStrip

4.7.1 ToolStripButton

Icons

Gibt man dem ToolStripButton mit der `Image` Eigenschaft ein Icon, erscheint es oft zu klein. Um dies zu verhindern sollte man die Eigenschaft `ImageScaling` auf `None` setzen.

Text

Damit der ToolStripButton statt eines Icons einen Text anzeigt, muss man die Eigenschaft `DisplayStyle` auf `ToolStripItemDisplayStyle.Text` setzen. Ansonsten zeigt der ToolStripButton nur das StandardIcon  .

4.8 ProgressBar und ToolStripProgressBar

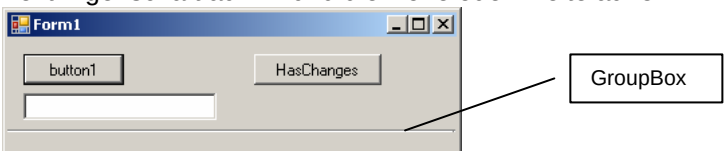
Ist ein längerer Vorgang in Ausführung und man weiss nicht, zu wie vielen Prozent der Vorgang schon abgelaufen ist, kann man eine ProgressBar in den **Marquee-Modus** versetzen (ein kurzer Balken läuft von links nach rechts durch), hat er das rechte Ende erreicht, beginnt er wieder links.

Beispiel:

```
// Millisekunden zwischen Animationsschritten setzen
ProgressBar1.MarqueeAnimationSpeed = 20;
ProgressBar1.Style = ProgressBarStyle.Marquee; // Animation starten
// Länger andauernder Vorgang
ProgressBar1.Style = ProgressBarStyle.Continuous; // Animation beenden
```

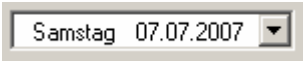
4.9 GroupBox

Die GroupBox kann auch als horizontaler oder vertikaler Balken verwendet werden. Dazu setzt man deren Text Eigenschaft auf "" und die Höhe oder Breite auf 5.



4.10 DateTimePicker

Die Eigenschaft `Format` kann man auf einen der vier Werte `Long`, `Short`, `Time` und `Custom` der Enumeration `DateTimePickerFormat` setzen. Setzt man sie auf `Custom`, kann man mit der Eigenschaft `CustomFormat` die Formatierung völlig frei wählen.



4.11 Validierung

4.11.1 Das Validating-Event

Um mit dem Validating-Event den User-Input eines Controls validieren zu können, muss dessen `CausesValidation` Eigenschaft auf `true` sein. Ebenfalls muss `CausesValidation` des nächsten Controls in der Tabulator-Reihenfolge auf `true` sein.
Angenommen `CausesValidation` von zwei TextBoxen ist auf `true` und der User springt mit TAB von der einen zur nächsten. Es erfolgt diese Sequenz von Ereignissen:

1. TextBox1.Leave
2. TextBox2.Enter
3. TextBox1.Validating
4. TextBox2.Validating
5. TextBox1.LostFocus
6. TextBox2.GotFocus

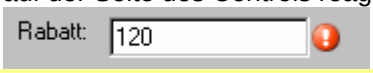
Ein möglicher ValidatingEventHandler für eine TextBox die ein Required Field ist:

```
private void txtName_Validating(object sender, CancelEventArgs e) {
   TextBox textBox;
   textBox = (TextBox)sender;
   if (textBox.Text.Length == 0) {
      MessageBox.Show("Sie müssen einen Namen eingeben");
      e.Cancel = true; // → txtName verliert Fokus nicht
   }
}
```

Eine bessere Möglichkeit auf eine ungültige Eingabe zu reagieren ist jedoch die Verwendung der Extenderanbieterkomponente ErrorProvider

4.11.2 Die Klasse ErrorProvider

Software die mit einer MessageBox auf Falscheingaben reagiert wird von Benutzern nicht sonderlich gemocht. Ein besserer Weg ist der ErrorProvider, welcher auf eine Falscheingabe mit einem blinkenden Icon auf der Seite des Controls reagiert. Ein ToolTip zeit die Fehlermeldung an.



Das obige Beispiel unter Verwendung des ErrorProviders:

```
private void txtName_Validating(object sender, CancelEventArgs e) {
   TextBox textBox;
   textBox = (TextBox)sender;
   if (textBox.Text.Length == 0) {
      ErrorProvider1.SetError(textBox, "Sie müssen Ihren Namen eingeben";
      e.Cancel = true; // → txtName verliert Fokus nicht
   }
   else {
      // falls die Falscheingabe korrigiert wurde
      ErrorProvider1.SetError(textBox, "");
   }
}
```

4.11.3 Formularvalidierung mit dem ErrorProvider

Indem wir im EventHandler `e.Cancel` auf `true` setzen, verhindern wir dass der User das Control verlassen kann. Lassen wir `e.Cancel = true` weg, müssen wir am Schluss beim Schliessen des Formulars prüfen ob der ErrorProvider für ein Control eine ErrorMessage hat die ungleich "" ist.

```
private void btnOK_Click(object sender, EventArgs e) {
    bool invalidInput = false;
    foreach (Control ctrl in this.Controls) {
        if (ErrorProvider1.GetError(ctrl).Length != 0){
            invalidInput = true;
            break; // springt aus der Schleife heraus
        }
    }
    if (invalidInput) MessageBox.Show("Immer noch ungültige Eingaben!");
    else this.Close();
}
```

4.12 Drag & Drop

4.12.1 Grundlegendes

Eine **Drag & Drop Operation** durchläuft folgende Phasen:

1. Der User klickt auf ein Control oder eine spezifische Region eines Controls und hält die linke Maustaste gedrückt. Bestimmte Informationen werden abgelegt und die Drag & Drop Operation beginnt.
2. Der Benutzer bewegt die Maus über ein anderes Control. Falls dieses den aktuellen Typ der verschobenen Informationen akzeptiert, wechselt der Mauszeiger zu einem speziellen Drag & Drop Icon. Andernfalls wird er zu einem durchgestrichenen Kreis.
3. Lässt der User die Maustaste los, erhält das Control die Information und entscheidet was damit zu tun ist. Die Operation sollte auch abzubrechen sein, indem man ESCAPE drückt (ohne die Maustaste loszulassen).

4.12.2 Drag & Drop implementieren

Die Eigenschaft `AllowDrop` des Zielsteuerelements muss auf `true` gesetzt werden.
Im `MouseDown`-EventHandler des Quellsteuerelements muss man dessen `DoDragDrop()` Methode ausführen.
`public DragDropEffects DoDragDrop(object data, DragDropEffects allowedEffects)`
`data` sind die zu transportierenden Daten.
Für `allowedEffects` ist meist `DragDropEffects.Copy` die richtige Wahl.
Im `DragEnter`-EventHandler der Zielsteuerelements `e.Effect` auf einen Wert setzen (meist `DragDropEffects.Copy`).
Im `DragDrop`-EventHandler des Ziel-Controls die transportierten Daten via `e.Data` holen. Mit `e.Data.GetDataPresent(typeof(`*`zieltyp`*`))` kann man ermitteln, ob die Daten in zieltyp konvertiert werden können.
Mit `(`*`zieltyp`*`)e.Data.GetData(typeof(`*`zieltyp`*`))` erhält man die Daten.
Beispiel: (DataGridViewRow)e.Data.GetData(typeof(DataGridViewRow));

4.13 Datenbindung

Die Klasse `System.Windows.Forms.Control` hat die Eigenschaft DataBindings welche vom Typ `ControlBindingsCollection` ist. Mit dieser Collection kann man auf alle Datenbindungen dieses Steuerelements zugreifen.

Beispiel:

```
textBox1.DataBindings.Add("Text", dsKunden.tbl_Kunde, "Name");
textBox1.DataBindings[0]; // Gibt die erste Datenbindung zurück (Klasse Binding)
textBox1.DataBindings["Text"]; // Gibt die Bindung an die Text-Eigenschaft zurück
```

4.13.1 Die Klasse Binding

Namespace: `System.Windows.Forms`
Mit den Methoden `ReadValue()` und `WriteValue()` kann man das Steuerelement zwingen, den Wert aus der Datenquelle zu lesen, resp. den Wert in die Datenquelle zu schreiben.

4.14 ReadOnly-Controls programmieren

```
public class ReadonlyControlHolder : Panel {
```

```
private bool readOnly = false;
public bool ReadOnly {
get {
return readOnly;
}
set {
if (readOnly != value) {
readOnly = value;
if (IsHandleCreated) {
SafeNativeMethods.EnableWindow(new HandleRef(this, this.Handle), !readOnly);
}
}
}
}
protected override void OnHandleCreated(EventArgs e) {
base.OnHandleCreated(e);
if (ReadOnly) {
SafeNativeMethods.EnableWindow(new HandleRef(this,this.Handle), !readOnly);
}
}
private class SafeNativeMethods {
[DllImport("user32")]
public static extern void EnableWindow(HandleRef hwnd, bool enabled);
}
}
```

5 Eigene Controls und Komponenten

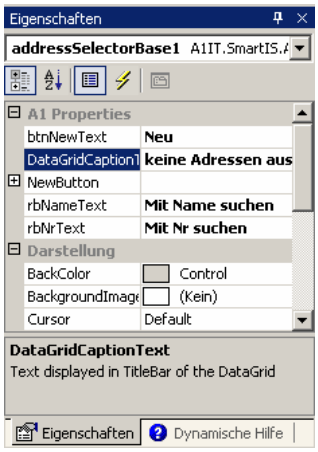
5.1 Allgemeines

5.1.1 Properties und das Eigenschaftenfenster

Properties von Komponenten und Steuerelementen werden standardmässig im Eigenschaftenfenster angezeigt, falls man sie aus der Toolbox in ein Formular übernimmt. Dies kann verhindert werden, indem man das Attribut `Browsable` verwendet.

Beispiel:

```
[Browsable(false)]
public string Name { ...
```



Mit dem Attribut `Category` lässt sich steuern in welcher Kategorie das Property angezeigt wird (falls man die Eigenschaften nach Kategorie sortiert.

Beispiel:

```
[Category("A1 Properties")]
public string Name { ...
```

Das Attribut `Description` definiert die Beschreibung die unterhalb des Eigenschaftenfensters erscheint, wenn die Eigenschaft ausgewählt ist.

Beispiel:

```
[Description("Text displayed in TitleBar of the DataGrid")]
public string DataGridCaption { ...
```

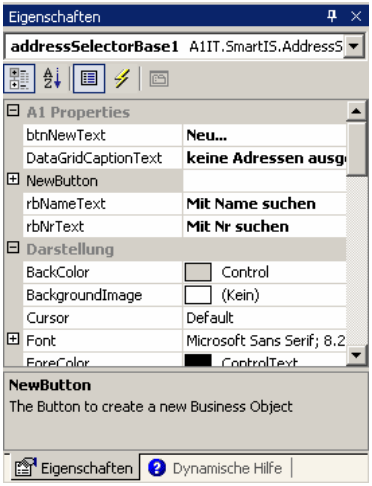
5.2 Ganze Controls als Properties

Bei selbst programmierten Steuerelementen kann man darin enthaltene Controls vollständig im Eigenschaftenfenster sichtbar und editierbar machen.

Beispiel:

```
protected System.Windows.FormsButton btnNew;
....
[Category("A1 Properties")]
public Button NewButton {
   get { return btnNew; }
}
```

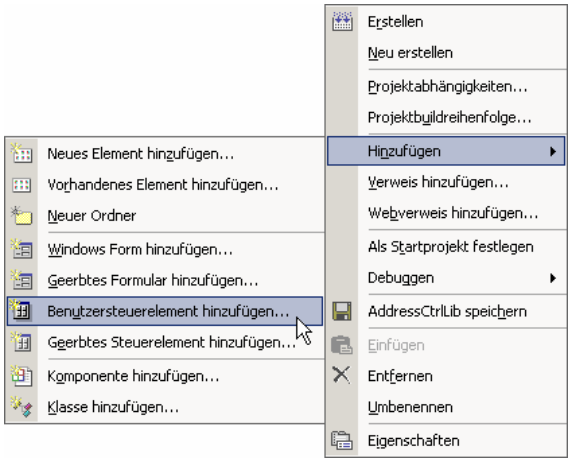
Benutzt man das selbstdefinierte Steuerelement erscheint NewButton als Eigenschaft des Steuerelements. Mit Klick auf das + werden alle Eigenschaften von NewButton sichtbar und editierbar.



5.3 Eigene Steuerelemente

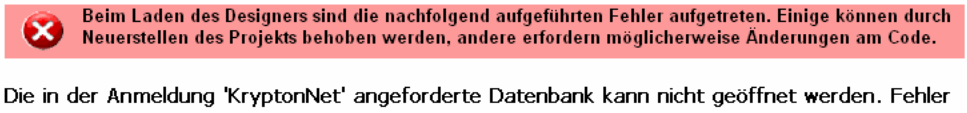
Um selbst ein Control von Grund auf zu programmieren leitet man von der Klasse `UserControl` aus dem Namespace System.Windows.Forms ab.
UserControls können in Windows-Anwendungen und Windows-Steuerelementbibliotheken definiert werden. Um nicht das Codegerüst selbst schreiben zu müssen verwendet man das Template "Benutzersteuerelement" (UserControl).

Eine weitere Möglichkeit für eigene Steuerelemente besteht darin, dass man ein Existierendes ableitet.



5.3.1 Vorsicht mit dem Control_Load-Event

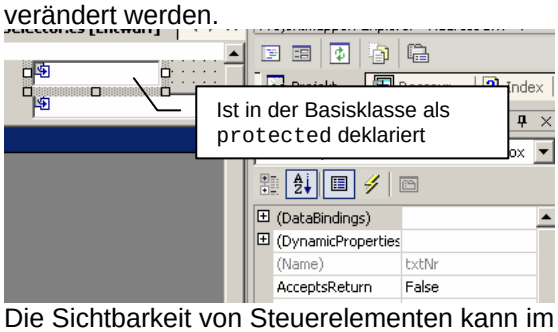
Bei Formularen scheint es kein Problem zu sein, Datenbankzugriffe im `Form_Load`-Event zu machen. Da die IDE jedoch bei Controls das `Load-Event` auslöst und darin enthaltenen Code ausführt, kommt es zu Problemen (das Formular kann nicht mehr angezeigt werden), wenn dabei eine Connection zu einer DB verwendet wird, die nicht zugänglich ist.

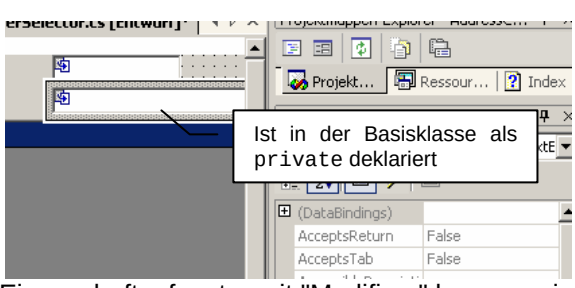


Eine Lösung ist das Schreiben einer Methode wie z.B. `DbInitialize()` in welcher zum erstenmal auf die DB zugegriffen wird. Als einbindender Programmierer eines Controls muss man darauf achten, zuerst die Connections richtig zu setzen, und danach diese Methode aufzurufen.

5.3.2 Eigene Steuerelemente ableiten

Leitet man ein zusammengesetztes Steuerelement ab, kann man die Eigenschaften von enthaltenen Controls die als `protected` deklariert sind problemlos Eigenschaftenfenster editieren sowie die Grösse und Position im Form Designer ändern. Enthaltene Controls die als `private` deklariert sind können nicht verändert werden.





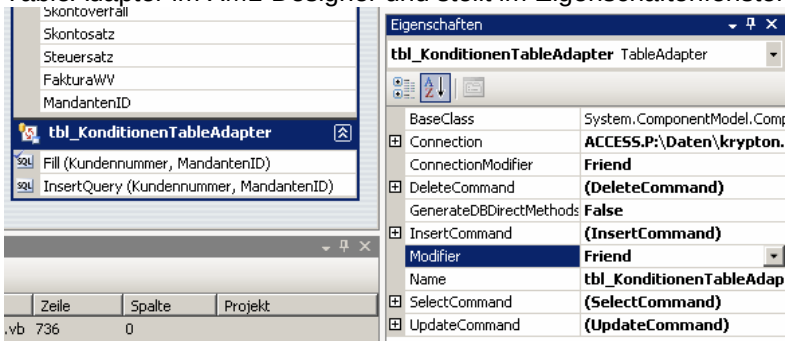
Die Sichtbarkeit von Steuerelementen kann im Eigenschaftenfenster mit "Modifiers" bequem visuell geändert werden.

5.3.3 Darstellung in Toolbox verhindern

Programmiert man eigene Controls als Steuerelementbibliothek, definiert man in der gleichen Assembly meist noch weitere Klassen die von `Component` abgeleitet sind (SubControls, DataSets, TableAdapter…). Diese werden dann in der Toolbox des einbindenden Projekts ebenfalls dargestellt. Dies lässt sich verhindern, indem man deren Sichtbarkeit auf `internal` setzt.

TableAdapter

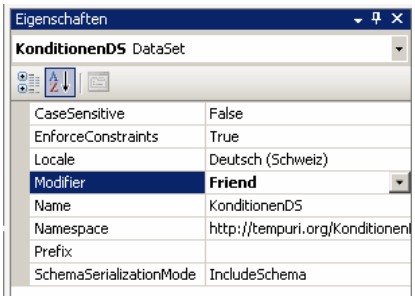
Um zu verhindern, dass TableAdapter ebenfalls in der Toolbox dargestellt werden, selektiert man den TableAdapter im XML-Designer und stellt im Eigenschaftenfenster Modifier auf `internal`.



Die Toolbox eines Projektes, welches eine Referenz auf KundenVW.dll hat. In der Toolbox benötigt wird jedoch nur KundenCtrl

Typisierte DataSets

Das typisierte DataSet im XML Designer darstellen, keine Tabelle und keinen TableAdapter auswählen, so dass im Eigenschaftenfenster die Eigenschaften des DataSets selbst dargestellt werden. Eigenschaft `Modifier` auf `internal` setzen. Die TableAdapter des DataSets dürfen aber keine höhere Sichtbarkeit besitzen als das DataSet.



Sub-Contols

Man setzt vor die Klassendefinition das Attribut `ToolboxItem(false)`

Beispiel:

```
[ToolboxItem(false)] public class KonditionenCtrl : UserControl
```

5.3.4 Designer-generierten Code steuern

Leitet man von `UserControl` ab, und fügt der Klasse neue Properties hinzu, erzeugt der Form-Designer für jedes dieser Properties eine Zeile im generierten Code. Properties welchen nicht im Eigenschaftenfenster ein Wert zugewiesen wird, werden dabei auf einen Standardwert gesetzt (`null` für Referenztypen). Dies kann zu unerwünschten Nebeneffekten führen.
Ob für ein Property Designer-generierter Code erzeugt werden soll, lässt sich mit dem Attribut `DesignerSerializationVisibility` steuern.

5.4 ComboBox ableiten

Setzt man die Eigenschaft `DrawMode` auf `DrawMode.OwnerDrawFixed` oder `DrawMode.OwnerDrawVariable` (falls die Items nicht die selbe Höhe haben) muss man das Zeichnen der Items selbst übernehmen. Dazu behandelt man das `DrawItem-Event`. Der Event-Handler übernimmt als zweiten Parameter eine Instanz von `DrawItemEventArgs`. `DrawItemEventArgs.State` ist vom Typ `DrawItemState` (eine Enumeration).

Werte der DrawItemState-Konstanten:

Name	Wert	Beschreibung
`Checked`	8	Das Element ist aktiviert. Dieser Wert wird nur von Menüsteuerelementen verwendet.
`ComboBoxEdit`	4096	Das Element ist der Bearbeitungsteil einer ComboBox.
`Default`	32	Das Element befindet sich im visuellen Standardzustand.
`Disabled`	4	Das Element ist nicht verfügbar.
`Focus`	16	Das Element besitzt den Fokus.
`Grayed`	2	Das Element ist grau unterlegt. Dieser Wert wird nur von Menüsteuerelementen verwendet.
`HotLight`	64	Das Element wird vorselektiert, d. h., das Element wird hervorgehoben, wenn der Mauszeiger darüber hinweg bewegt wird.
`Inactive`	128	Das Element ist inaktiv.
`NoAccelerator`	256	Das Element wird ohne Zugriffstaste angezeigt.
`NoFocusRect`	512	Das Element wird ohne den visuellen Hinweis angezeigt, der angibt, dass es den Fokus besitzt.
`None`	0	Das Element hat derzeit keinen Zustand.
`Selected`	1	Das Element ist ausgewählt.

Diese Werte kommen in Kombination vor. Gibt man im Event-Handler `e.State.ToString()` aus, erhält man Werte wie `"Selected, Focus, ComboBoxEdit"`.

Beispiel für einen DrawItem-EventHandler:

```
protected void drawItem(object sender, DrawItemEventArgs e) {
	if (_displayMembers == null) {
		return;
	}
	Brush brush = Brushes.Black;
	e.DrawBackground();
	// prüfen ob Selected-Flag gesetzt
	if ((e.State & DrawItemState.Selected) == DrawItemState.Selected) {
		brush = Brushes.White;
	}
	// prüfen ob Focus-Flag gesetzt
	if ((e.State & DrawItemState.Focus) == DrawItemState.Focus) {
		e.DrawFocusRectangle();
	}
	string[] strings = itemStrings(e);
	int startX = 0;
	for (int i = 0; i < _displayMembers.Length; i++) {
		e.Graphics.DrawString(strings[i],
			this.Font,
			brush,
			new RectangleF(e.Bounds.X + startX,
				e.Bounds.Y,
				_displayMemberWidths[i],
				e.Bounds.Height));
		startX += _displayMemberWidths[i];
	}
}
```

6 Konsolenanwendungen

6.1 Programmabbruch

`System.Environment.Exit(retCode);`

7 ADO.NET

7.1 Datenbindung

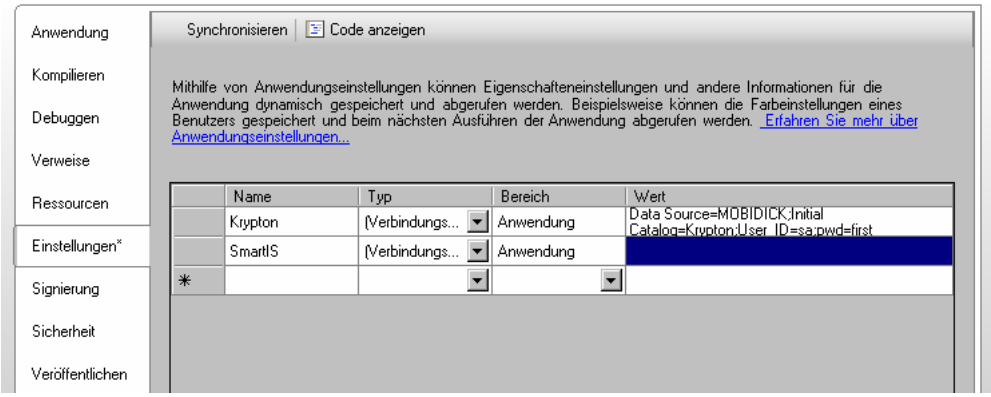
7.1.1 Datenquellen und ihre Interfaces

	IList	IListSource	ICollection	IEnumerable	IBindingList	IBindingListView	ITypedList
BindingSource	X		X	X	X	X	X
DataView	X		X	X	X	X	X
DataTable		X					
ArrayList	X		X	X			
Array	X		X	X			

7.2 ConnectionStrings

7.2.1 ConnectionStrings in App.config verwalten

ConnectionStrings werden am besten über Projekteigenschaften | Einstellungen im `App.config` gespeichert. Man sollte darauf achten, dass der Typ "`(Verbindungszeichenfolge)`" ist, und der Bereich "`Anwendung`".



Der Eintrag Krypton erzeugt folgenden Eintrag im `App.config`:

```
<connectionStrings>
    <add
        name="Test2.My.MySettings.Krypton"
        connectionString="Data Source=MOBIDICK;Initial Catalog=Krypton;User ID=sa;pwd=secret"
        providerName="System.Data.SqlClient"
    />
</connectionStrings>
```

`Test2` ist der Stammnamespace des Projekts. `My.MySettings` ist ein Subnamespace. Krypton ist der Name des ConnectionStrings in der App.config.

Auf diesen ConnectionString kann man im Code folgendermassen zugreifen:

```
Imports System.Configuration
...
ConfigurationManager.ConnectionStrings("Test2.My.MySettings.Krypton").ConnectionString
```

WICHTIG: Verweis auf `System.configuration.dll` setzen!!!
Unter C# würde der Eintrag im `App.config` so aussehen:

```
<connectionStrings>
    <add
        name="Test2.Properties.Settings.Krypton"
        connectionString="Data Source=MOBIDICK;Initial Catalog=Krypton;Integrated Security=True"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

7.2.2 Connection Timeout

Bei SQL Server ConnectionStrings kann das Connection Timeout mit "`Connect Timeout=`*`sekunden`*`;`" spezifiziert werden.

7.3 Commands

7.3.1 OleDbCommand

Ruft man `OleDbCommand.ExecuteScalar()` auf und es wird in der DB kein Wert gefunden, da kein entsprechender Datensatz existiert, ist der Rückgabewert `null` (nicht `System.DbNull.Value`).

7.4 Concurrency

ADO.NET reagiert, wenn ein Client Daten in die DB zurück schreibt, die während der Bearbeitung durch einen anderen Client geändert wurden. Es wird eine `System.Data.DBConcurrencyException` geworfen.
Siehe auch: MSDN-Artikel „Einführung in die Datenparallelität in ADO.NET“.

7.5 DataSets

7.5.1 Typisierte DataSets

Beim Erstellen eines typisierten DataSets wird für jede Tabelle ein Klasse von `System.Data.DataTable` abgeleitet. Ebenfalls wird für jede Tabelle eine Klasse von `System.Data.DataRow` abgeleitet. Diese DataRows haben für jedes Feld eine Methode `bool Is<Feldname>Null()` mit welcher geprüft werden kann, ob der Wert `DBNull` ist. Diese Methode ist notwendig, da es zu einer Exception kommt, wenn man darauf zugreift und der Wert DBNull ist.
Zusätzlich gibt es für jedes Feld eine Methode `void Set<Feldname>Null()`.

Die Struktur typisierter DataSets kann geändert werden, indem man im Projektmappen-Explorer auf <DataSetName>.xsd doppelklickt. Es öffnet sich der XML-Schema Designer und man kann Tabellen löschen, Spaltennamen ändern usw. Soll eine neue Tabelle zu einem DataSet hinzugefügt werden, macht man das am besten über den DataAdapter und den Hyperlink DataSet generieren.

7.5.2 DataSet Schema bearbeiten

Das Schema einer DataTable kann problemlos im XML-Designer um eine neue Spalte ergänzt werden, auch wenn diese in der Datenbank nicht existiert. Manchmal kann es nützlich sein, in der Applikation eine weitere Spalte zu haben mit deren Werten man arbeitet, die aber nicht in der Datenbank gespeichert werden müssen.
ACHTUNG! Schreibt man Werte in die neu kreierte Spalte, ändert sich die `RowState` Eigenschaft der betroffenen Zeile und der Adapter aktualisiert diese Zeile beim nächsten Aufruf von `Update()` in der Datenbank, auch wenn sich nur Daten in der Spalte geändert haben die nicht in der Datenbank existiert.

7.5.3 Prüfen ob sich Daten geändert haben

DataTable.`GetChanges()` liefert häufig nur verlässliche Resultate, wenn man zuvor `Form.Validate()` resp. `UserControl.Validate()` und `DataTable.EndEdit()` aufruft. Diese beiden Methoden sollten auch vor `TableAdapter.Update()` aufgerufen werden, da sonst der Wert aus der TextBox in welcher noch der Cursor ist, nicht übernommen werden.

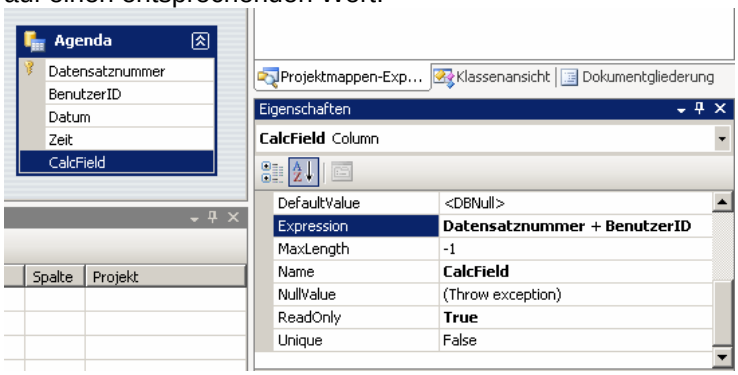
7.6 DataTables

7.6.1 Vorsicht mit Standardwerten

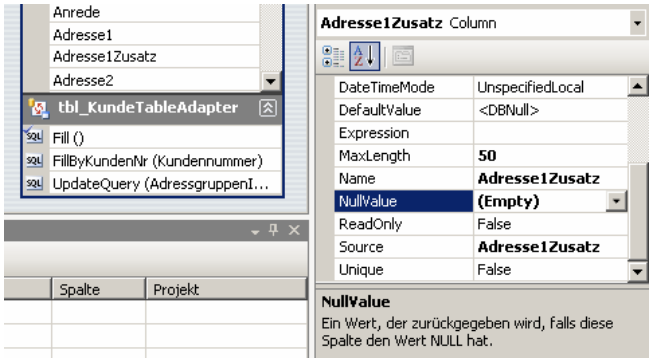
Zumindest bei Access scheinen die in der DB definierten Standardwerte beim generieren eines typsisierten DataSets nicht automatisch übernommen zu werden. Daher wird beim hinzufügen eines neuen Datensatzes in diese Felder `DbNull` eingefügt. Die Standardwerte müssen also manuell beim der DataTable gesetzt werden.

7.6.2 Berechnete Felder

Bei der datenbankunabhängigen Programmierung sind String-Konkatenierungen oft problematisch, da sich die Syntax je nach SQL Dialekt unterscheidet. Eine Möglichkeit, dies zu umgehen, ist in der DataTable ein berechnetes Feld einzufügen. Dazu setzt man die `Expression` Eigenschaft manuell hinzugefügten Feldes auf einen entsprechenden Wert.



7.6.3 Umgang mit DbNull



7.6.4 Load() Methode

Die `Load()` Methode bietet eine Alternative zum Füllen via DataAdapter oder TableAdapter. Sie übernimmt ein Argument vom Typ `IDataReader`, dieses Interface wird von allen DataReadern implementiert. Mann kann ihr also einen OleDbDataReader, SqlDataReader, MySqlDataReader usw. übergeben.

7.6.5 Select() Methode

Gibt einen Array von `DataRows` zurück. Je nach Überladung kann man ein Filterkriterium (`WHERE`-Klausel ohne "WHERE"-Schlüsselwort) und eine Sortierreihenfolge angeben. Handelt es sich um eine typisierte DataTable, kann der Array zu einem Array vom Typ der typisierten DataRow konvertiert werden.

Beispiel:

```
DataRow[] arrUntypedRows;
arrUntypedRows = dsKunden.tbl_Kunde.Select("SchwarzeListe = true", "Kundennummer");
KundenDS.tbl_KundeRow[] arrTypedRows = (KundenDS.tbl_KundeRow[])arrUntypedRows;
```

7.6.6 Durch Iteration bestimmte Zeilen löschen

Falsch:

```
for (int i = 0; i < dsBelegpositionen.tbl_Belegzeile.Rows.Count; i++) {
	BelegpositionenCtrlDS.tbl_BelegzeileRow row;
	row = (BelegpositionenCtrlDS.tbl_BelegzeileRow)dsBelegpositionen.tbl_Belegzeile.Rows[i];
	if (row.RowState != DataRowState.Deleted && row.Zeilentyp == "Stklst") {
		row.Delete();
	}
}
```

Obiges Beispiel funktioniert nicht, da beim Löschen einer Zeile, die ZeilenIndizes der nachfolgenden Zeilen sofort nachrücken. Kommen zwei Zeilen mit Zeilentyp "Stklst", wird die zweite nicht gelöscht.

Richtig:

```
int i = 0;
while (i < dsBelegpositionen.tbl_Belegzeile.Rows.Count) {
	if (dsBelegpositionen.tbl_Belegzeile[i].RowState != DataRowState.Deleted
		&& dsBelegpositionen.tbl_Belegzeile[i].Zeilentyp == "Stklst") {
		dsBelegpositionen.tbl_Belegzeile[i].Delete();
	}
	else {
		i++;
	}
}
```

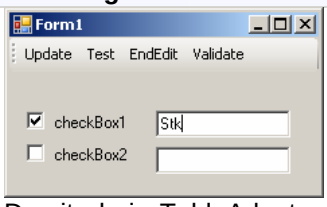
Die Iterationsvariable darf nur erhöht werden, wenn die vorherige Zeile nicht gelöscht wurde.

7.7 DataRows und Datenbindung

DataRows haben die Eigenschaft RowState vom Typ DataRowState (eine Enumeration).
DataRowState kann die folgenden Werte haben:

Membername	Beschreibung
Added	Die Zeile wurde einer **DataRowCollection**-Klasse hinzugefügt, und AcceptChanges wurde nicht aufgerufen.
Deleted	Die Zeile wurde mit der **Delete**-Methode von **DataRow** gelöscht.
Detached	Die Zeile wurde erstellt, ist jedoch nicht Teil einer **DataRowCollection**-Klasse. Eine **DataRow**-Klasse befindet sich in diesem Zustand, wenn sie unmittelbar nach ihrer Erstellung noch keiner Auflistung hinzugefügt wurde oder wenn sie aus einer Auflistung entfernt wurde.

Modified	Die Zeile wurde geändert, und **AcceptChanges** wurde nicht aufgerufen.
Unchanged	Die Zeile wurde nach dem letzten Aufruf von **AcceptChanges** nicht geändert.



Damit bei TableAdapter.Update() die Änderungen in die DB zurückgeschrieben werden, muss DataRow.RowState den Wert Modified aufweisen. Dies ist jedoch erst nach einem Aufruf von DataRow.EndEdit() der Fall.
Damit nach dem Editieren einer TextBox, `DataRow.RowState` den Wert `Modified` enthält, muss **einer** der folgenden Vorgänge ablaufen:

- Der User verlässt die TextBox und es wird `DataRow.EndEdit()` aufgerufen.
- Es wird `Validate()` (des Forms oder Controls) und danach `DataRow.EndEdit()` aufgerufen[1].

Sowohl Form als auch UserControl erben die Methode Validate() von ContainerControl. In der .NET-Framework 2.0 Dokumentation steht darüber folgendes:
Überladen. Überprüft den Wert des Steuerelements, das den Fokus verliert, indem das Validating-Ereignis und das Validated-Ereignis in dieser Reihenfolge ausgelöst werden. (Von ContainerControl geerbt.)

7.8 Untypisierte DataTables

7.8.1 DbNull

Um herauszufinden, ob ein bestimmter Wert DbNull ist, kann man ihn mit `System.DbNull.Value` vergleichen.
Beispiel:

```
// dieser bool'sche Ausdruck evaluiert sich zu true, falls das entspr. Feld in der DB DbNull ist
dtAtrikel.Rows[0]["ArtikelSerieNr"] == System.DBNull.Value
```

7.9 DataViews

Eine DataTable hat immer bereits eine DataView, diese ist durch das Property `DefaultView` abrufbar.
Will man eine DataView filtern, so dass sie nur noch Datensätze anzeigt die in einer bestimmten Spalte DBNULL haben, lässt sich dies mit `RowFilter = "<Spaltenname> IS NULL"` erreichen.

7.10 DataAdapter

Verändert man einen Parameter eines DataAdapters und benutzt ihn um eine DataTable zu füllen, wird diese nicht automatisch geleert, die neuen DataRows werden hinzugefügt.

7.11 DataReader

Um zu prüfen, ob ein Feld DbNull ist, muss man die Methode `IsDbNull(`*`Feldposition`*`)` verwenden. Da man mit Feldpositionen arbeiten muss, verliert man aber an Flexibilität und die Fehlerwahrscheinlichkeit steigt.
Tipp: Methode `getOrdinal(`*`Feldname`*`)` verwenden.
Beispiel: Testen ob das Feld Name DbNull ist:

```
if (reader.IsDbNull(reader.getOrdinal("Name"))) {…
```

7.12 SqlParameter

7.12.1 Wichtige Eigenschaften

`public virtual string` **`SourceColumn`** `{get, set}`
Der Name der Spalte in der DataTable. Wird ein SqlCommand als UpdateCommand eines SqlDataAdapters verwendet, holt sich der DataAdapter für die betroffe DataRow diesen Wert, weist ihn der Eigenschaft `Value` des Parameters zu und führt das Command aus.

[1] In diesem Fall ist es nicht notwendig, dass der User die TextBox verlässt.

7.13 Weitere Vorgehensweise und Techniken

7.13.1 Typisierte DataTables und DataRows erweitern

Klickt man im Projektmappen-Explorer mit der rechten Maustaste auf den Code-File eines DataSets und wählt Code anzeigen öffnet sich ein Code-File in welchem man dem DataSet und den darin enthaltenen DataTables und DataRows weitere Member hinzufügen kann. Partielle Klassen macheb es möglich.
Dabei ist es wichtig, beim Erweitern einer DataRow-Klasse, diese partielle Klasse innerhalb der umschliessenden partiellen DataSet-Klasse zu definieren.

```
namespace Krypton.Net {

    public partial class BelegpositionenCtrlDS {

        public partial class tbl_BelegzeileRow {
            public void TestMethod() { // neu hinzugefügte Methode

            }
        }
    }

}
```

7.14 Mit ADO.NET auf Excel zugreifen

7.14.1 Zugriff auf eine Excel XP Datei mit OleDb

ConnectionString
Provider=Microsoft.JET.OLEDB.4.0;Data Source=<Pfad>;Extended Properties=EXCEL 8.0

SelectCommand.CommandText
```
SELECT * FROM [<Tabellenname>$]
```

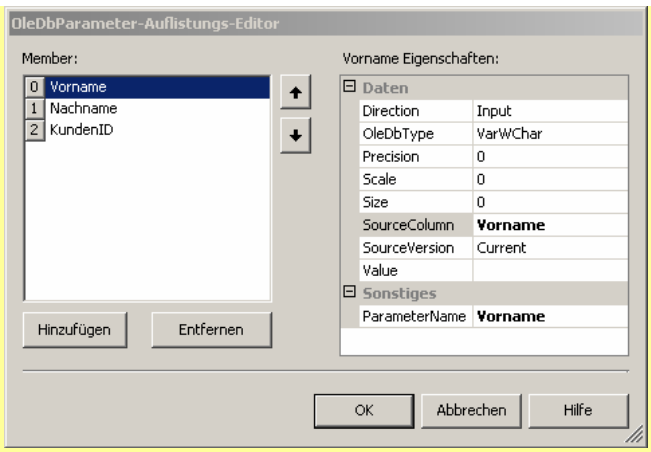
Der Versuch, eine OleDbConnection zu einem Excel File in der IDE mit dem Server Explorer oder dem Assistenten zu erzeugen schlug bisher fehl. Der ConnectionString und die SQL Statements müssen also manuell zugewiesen werden.

Konkretes Beispiel
Zugriff auf die Arbeitsmappe Daten.xls welche im gleichen Verzeichnis wie die Applikation liegt und eine Tabelle Namens Kunden hat:

ConnectionString:
```
Provider=Microsoft.JET.OLEDB.4.0;
Data Source=Daten.xls;
Extended Properties=EXCEL 8.0
```
SelectCommand.CommandText:
```
SELECT * FROM [Kunden$]
```

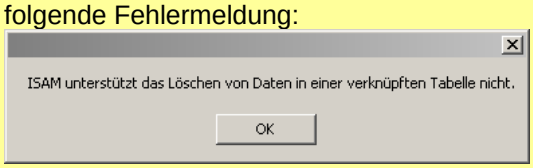
InsertCommand.CommandText:
```
INSERT INTO [Kunden$] (KundenID, Vorname, Nachname) VALUES (?,?,?)
```
Für den ersten Parameter habe ich den Typ OleDbType.Integer verwendet, für die anderen beiden OleDbType.VarWChar. Die Parameter lassen sich problemlos visuell im Eigenschaftenfenster des OleDbDataAdapters erzeugen. OleDbType.VarWChar wird von VS .NET automatisch als Standard vorgeschlagen. Wichtig ist das Setzen der Eigenschaft SourceColumn auf den Spaltennamen in der Excel Tabelle!

UpdateCommand.CommandText:
```
UPDATE [Kunden$] SET Vorname = ?, Nachname = ? WHERE KundenID = ?
```
Da OleDbParameter im SQL Statement nicht mit Namen aufgeführt werden können muss man bei der ParameterCollection des Commands die richtige Reihenfolge beachten!



DeleteCommand.CommandText:
```
DELETE FROM [Kunden$] WHERE KundenID = ?
```

Das Laden, Einfügen und Updaten von Datensätzen funktionierte problemlos. Beim Löschen erschien folgende Fehlermeldung:



Die Tabelle war aber mit keiner anderen verknüpft, ich habe eine neue Excel Datei für diesen Versuch erstellt.

Die eckigen Klammern beim Tabellenamen sind auch dann notwendig, wenn dieser keine Leerzeichen oder Sonderzeichen enthält!

7.15 Arbeiten mit Access

7.15.1 Access SQL-Syntax

7.15.1.1 String-Konkatenierungen die mit einer Zahl beginnen

```
SELECT STR(Kundennummer) + ': ' + Name AS KundeNrName…
```

Ist Kundennummer kein numerischer Typ, funktioniert obige Zeile nur dank der STR() Funktion.

7.15.1.2 Aufgepasst bei String-Konkatenierungen mit '+'

```
SELECT LastName + ' ' + FirstName AS FullName FROM…
```

Ist FirstName oder LastName DBNULL so wird FullName ebenfalls DBNULL (bei Konkatenierung mit '&' ist dies nicht der Fall)

7.15.1.3 Prüfen ob ein bestimmter Wert DbNull ist

```
SELECT     COUNT(*) AS Anzahl
FROM       tbl_Inventarliste
WHERE      (Artikelnummer = ?) AND (Menge IS NULL)
```

Die einzige mir bekannte Möglichkeit, ist zu zählen wieviele Datensätze (mit entsprechendem Schlüssel), in der betreffenden Zeile DbNull haben.

7.15.1.4 Datumsberechnungen

```
SELECT PersNr, Name, Vorname, GebDat, DATEDIFF('yyyy', GebDat, NOW()) AS [Alter]
FROM Personal
ORDER BY DATEDIFF('yyyy', GebDat, NOW())
```

`DATEDIFF( )` verlangt als ersten Parameter das zu berechnende Intervall. Dafür gibt es folgende Werte:

Intervall-String:	**Beschreibung:**
yyyy	Jahre
q	Quartale
m	Monate
y	Tag des Jahres
d	Tage

w	Wochentage
ww	Wochen
h	Stunden
n	Minuten
s	Sekunden

7.15.1.5 Teile aus einem Datum extrahieren

Dafür gibt es die Funktion `DATEPART(<datumsteil>, <datum>)`.
Für `<datumsteil>` können die gleichen Intervall-Strings verwendet werden wie bei `DATEDIFF`.

7.15.1.6 Vorsicht bei den ganzzahligen Typen von Access

Ein `Integer` in Access entspricht nicht einem `int` in C#. Bei Access handelt es sich um eine 16-Bit Ganzzahl (Wertebereich: -32'768 bis -32'767), dies entspricht in C# einem `short`, resp. `System.Int16`.

7.15.1.7 Zuletzt eingefügter AutoWert nachschlagen

```
cmdAutoIncrement.CommandText = "SELECT @@IDENTITY";
```

Zwischen dem INSERT-Statement und "SELECT @@IDENTITY" darf die Connection nicht geschlossen werden.

7.15.1.8 Timestamp mit OleDbCommand in DB schreiben

Beispiel:

```
cmdInsert.CommandText = String.Format(
   "INSERT INTO tbl_Errors (SurveyId, Page, [Error], OccurenceTime) VALUES ({0}, '{1}', '{2}', ?)",
   Session["SurveyId"], Request["aspxerrorpath"], ex.Message);
cmdInsert.Parameters.Add("prmTimestamp", OleDbType.Date);
cmdInsert.Parameters["prmTimestamp"].Value = DateTime.Now;
```

Es gibt auch den Parametertyp `OleDbType.DBTimeStamp` doch damit funktioniert es nicht.

7.16 Providerfabriken

Die Klasse `DbProviderFactory` ermöglicht, Abkömmlinge von abstrakten Klassen wie DbConnection, DbCommand, DbDataAdapter usw zu erzeugen. Dies sind OleDbConnection, OdbcConnection, OleDbCommand usw. Damit kann man Code schreiben, welcher unabhängig von einer spezifischen Datenbank ist.
Eine DbProviderFactory instanziiert man über die statische Methode `GetFactory()` der Klasse `DbProviderFactories`. Sie übernimmt ein Argument `InvariantName` vom Typ `String`. Jedem Datenprovider ist ein solcher String zugeordnet. Eine Tabelle mit allen Providern und deren InvariantNames erhält man über die Methode `DbProviderFactories.GetFactoryClasses()`.

Name	Description	InvariantName	AssemblyQualifie
SQL Server CE Data Provider	.NET Framework Data Provider for Microsoft SQL Server 2005 Mobile Edition	Microsoft.SqlServerCe.Client	Microsoft.SqlServ
Odbc Data Provider	.Net Framework Data Provider for Odbc	System.Data.Odbc	System.Data.Odb
OleDb Data Provider	.Net Framework Data Provider for OleDb	System.Data.OleDb	System.Data.OleD
OracleClient Data Provider	.Net Framework Data Provider for Oracle	System.Data.OracleClient	System.Data.Orac
SqlClient Data Provider	.Net Framework Data Provider for SqlServer	System.Data.SqlClient	System.Data.SqlC

7.16.1 DbCommandBuilder

Mit `DbProviderFactory.CreateCommandBuilder()` erhält man einen providerspezifischen CommandBuilder. Nachdem man dessen `DataAdapter` Eigenschaft einen DbDataAdapter mit initialisiertem SelectCommand zugewiesen hat, stehen die Methoden `getInsertCommand()`, `getDeleteCommand()` und `getUpdateCommand()` zur Verfügung.

7.16.2 Vorsicht bei SQL Server

Da der SQL Server in CommandText benannte Parameter (z.B. @Name) benötigt und unbenannte Parameter, also ? nicht akzeptiert ist manchmal noch etwas Handarbeit notwendig. Am besten man schreibt den CommandText für das SelectCommand mit benannten Parametern und ersetzt diese durch ? falls der Povider nicht System.Data.SqlClient ist.

7.16.3 Zugriff auf MySQL via ODBC und MyODBC-Connector

Beispiel für ConnectionString:

```
"DRIVER={MySQL ODBC 3.51 Driver};SERVER=localhost;DATABASE=test;UID=venu;PASSWORD=venu;OPTION=3";
```

MySQL Connector/ODBC 3.51 kann hier bezogen werden:
http://dev.mysql.com/downloads/connector/odbc/3.51.html

7.16.4 Probleme und Lösungen bei der datenbankunabhängigen Programmierung

String Concatenation

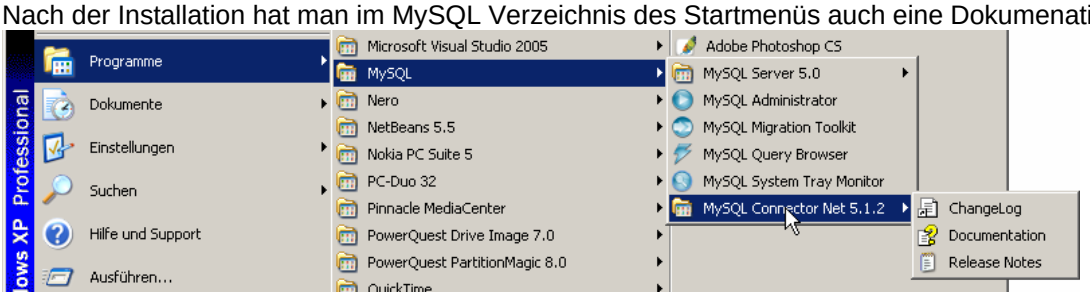
Access und SQL Server:
```
SELECT Name + ' ' + Vorname AS Bezeichnung FROM tbl_Mitarbeiter;
```
MySQL:
```
SELECT CONCAT(Name, ' ', Vorname) AS Bezeichnung FROM tbl_Mitarbeiter;
```
ANSI SQL 92:
```
SELECT Name || ' ' || Vorname AS Bezeichung FROM tbl_Mitarbeiter;
```

Mögliche Lösungen

- In der Datenbank eine View definieren, welche die Konkatenierung vornimmt.
- In der DataTable ein berechnetes Feld definieren.

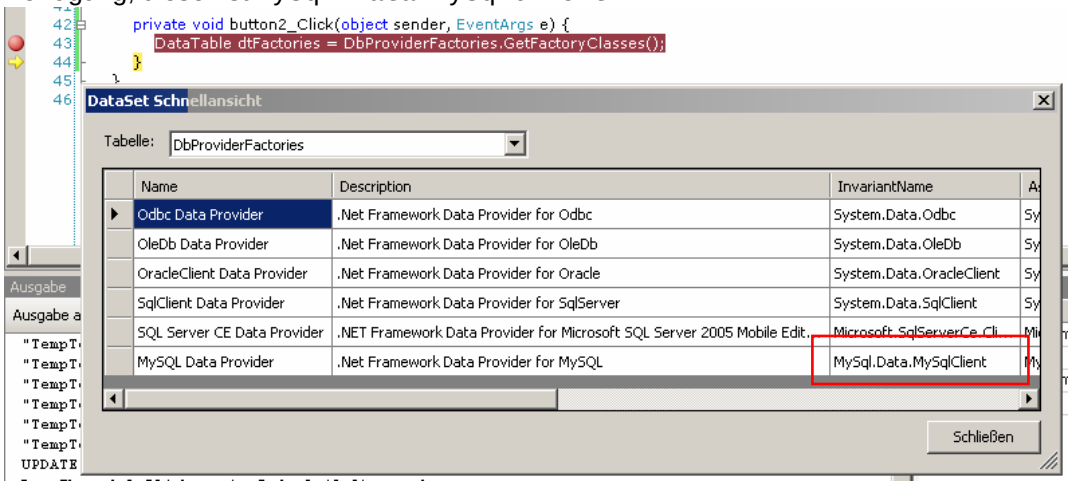
7.17 Zugriff auf MySQL

Den neusten ADO.NET Treiber kann man hier herunterladen:
http://dev.mysql.com/downloads/connector/net/5.1.html (Version 5.0.xxx macht Probleme)
Nach der Installation hat man im MySQL Verzeichnis des Startmenüs auch eine Dokumenation zum Treiber:



7.17.1 Generischer Zugriff mit DbProviderFactories

Nach der Installation des Treibers, steht für die Klasse DbProviderFactories ein neuer InvariantName zur Verfügung, dieser ist `MySql.Data.MySqlClient`:



Beispiel:

```
14  public partial class Form1 : Form {
15      DbProviderFactory dbFactory;
16      DbConnection conn;
17      DbDataAdapter daArtikel;
18
19      private void Form1_Load(object sender, EventArgs e) {
20          dbFactory = DbProviderFactories.GetFactory("MySql.Data.MySqlClient");
21          conn = dbFactory.CreateConnection();
22          conn.ConnectionString = "server=localhost;user id=root; pwd=first; database=krypton";
23          daArtikel = dbFactory.CreateDataAdapter();
24          daArtikel.SelectCommand = dbFactory.CreateCommand();
25          daArtikel.SelectCommand.Connection = conn;
26          daArtikel.SelectCommand.CommandText = "SELECT Artikelnummer, Artikelbezeichnung1 FROM tbl_artikel;";
27          DbCommandBuilder cmdBuilder = dbFactory.CreateCommandBuilder();
28          cmdBuilder.DataAdapter = daArtikel;
29          daArtikel.UpdateCommand = cmdBuilder.GetUpdateCommand();
30          daArtikel.DeleteCommand = cmdBuilder.GetDeleteCommand();
31          daArtikel.InsertCommand = cmdBuilder.GetInsertCommand();
32          daArtikel.Fill(dataSet11.tbl_artikel);
33      }
34
35      public Form1() {
36          InitializeComponent();
37      }
38
39      private void button1_Click(object sender, EventArgs e) {
40          daArtikel.Update(dataSet11.tbl_artikel);
41      }
```

Die vom `DbCommandBuilder` (hinter welchem ein `MySqlCommandBuilder` steckte), erzeugten Commands sehen wie folgt aus:

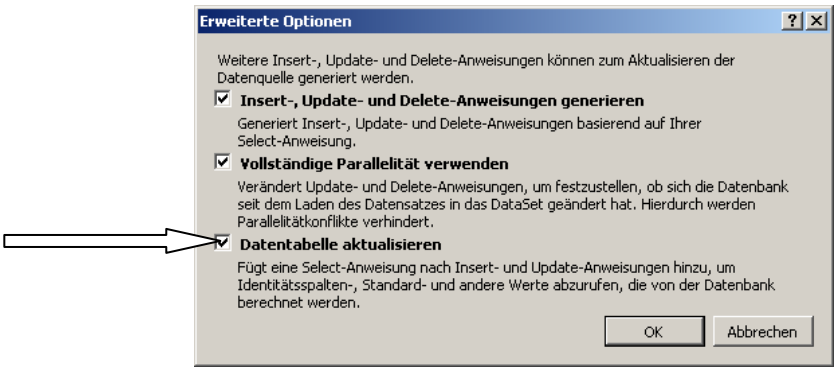
```
UpdateCommand:
UPDATE `krypton`.`tbl_artikel` SET `Artikelnummer` = ?p1, `Artikelbezeichnung1` = ?p2 WHERE
((`Artikelnummer` = ?p3))
```

MySQL hat also benannte Parameter, die im Gegensatz zu SQL Server nicht mit "@" beginnen, sondern mit "?".

```
DeleteCommand:
DELETE FROM `krypton`.`tbl_artikel` WHERE ((`Artikelnummer` = ?p1))

InsertCommand:
INSERT INTO `krypton`.`tbl_artikel` (`Artikelnummer`, `Artikelbezeichnung1`) VALUES (?p1, ?p2)
```

Weitere Details über den ConnectionString entnimmt man am besten der Dokumentation zum Treiber (standardmässig unter Start | Programme | MySQL | MySQL Connector .Net XXX).

7.17.2 AUTO_INCREMENT Spalten



Arbeitet man mit TableAdapter, kann man unter Erweiterte Optionen die CheckBox "Datentabelle aktualisieren" aktivieren, wodurch sich der TableAdapter um die AUTO_INCREMENT-Wert kümmert. Leider hat MySQL Connector .NET 5.X die Angewohnheit bei den generierten SQL-Anweisungen vor den Tabellennamen jeweils den Datenbanknamen hinzuzufügen. Dies verhindert einen Wechsel zwischen Test-Datenbank und produktiver Datenbank. Entfernt man diese Datenbanknamen aus den SQL-Statements, werden auch die (unsichtbaren) SELECT-Statements entfernt, welche für das korrekte Übernehmen der AUTO_INCREMENT-Spalten sorgt.

1. Lösung:

Nach TableAdapter.Update() jeweils wieder einen Fill() machen.

2. Lösung:

TableAdapter.Update() für jede einzelne Zeile aufrufen. Falls der RowState einer Zeile DataRowState.Added ist, danachh die MySQL-Funktion LAST_INSERT_ID() aufrufen um den vergebenen AUTO_INCREMENT-Wert zu ermitteln. Dabei stösst man aber auf eine weitere Herausforderung: Der Aufruf von TableAdapter.Update() für eine Zeile mit DataRowState.Deleted löscht diese Zeile aus der DataTable. Bei der Iteration via `foreach` oder `for` führt dies zu einer `InvalidOperationException`. Lösung: do ... while Schlaufe verwenden und die Iterationsvalriable i nur dann inkrementieren, wenn der RowState der aktuellen Zeile ungleich DataRowState.Deleted ist.

Beispiel:

```
private void save() {
```

```
    Validate();
    dgvBelegpositionen.EndEdit();
    try {
        MySqlCommand cmdAutoIncrement = _mySqlCnnKrypton.CreateCommand();
        cmdAutoIncrement.CommandText = "SELECT LAST_INSERT_ID();";
        mySqlCnnActivator.Activate();
        int i = 0;
        do {
            BelegpositionenCtrlDS.tbl_BelegzeileRow row = dsBelegpositionen.tbl_Belegzeile[i];
            DataRowState state = row.RowState; // DataRowState zwischenspeichern
            taBelegzeile.Update(row);
            // falls Datensatz eingefügt: korrekter AutoIncrement-Wert einfügen
            if (state == DataRowState.Added) {
                row.Datensatznummer = Convert.ToInt32(cmdAutoIncrement.ExecuteScalar());
            }
            // falls Datensatz gelöscht, rutschen die Indizes der folgenden Datensätze nach,
            // i++ erübrigt sich
            if (state != DataRowState.Deleted) {
                i++;
            }

        } while (i < dsBelegpositionen.tbl_Belegzeile.Rows.Count);
        dsBelegpositionen.tbl_Belegzeile.AcceptChanges();
    }
    catch (MySqlException ex) {
        MessageBox.Show(ex.Message, "Krypton.Net", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
    finally {
        mySqlCnnActivator.Deactivate();
    }
}
```

Macht man einen INSERT in eine Tabelle deren Primärschlüssel eine AUTO_INCREMENT-Spalte ist, muss man herausfinden können, welcher AUTO_INCREMENT-Wert vergeben wurde.
Dazu bietet MySQL die Funktion `LAST_INSERT_ID()`.

Beispiel:

```
MySqlCommand cmdInsert = _mySqlCnnKrypton.CreateCommand();
cmdInsert.CommandText = String.Format(
   "INSERT INTO tbl_Bestellungskopf (Belegsnummer, Druckstatus) VALUES ({0}, -1);", _belegsNr);
mySqlCnnKrypton.Open();
int recordsAffected = cmdInsert.ExecuteNonQuery();
MySqlCommand cmdInsertId = _mySqlCnnKrypton.CreateCommand();
cmdInsertId.CommandText = "SELECT LAST_INSERT_ID()";
int autoIncrement = Convert.ToInt32(cmdInsertId.ExecuteScalar());
// newRow ist die DataRow, welche in MySQL geschrieben wurde
// Bestellnummer ist der Primärschlüssel welche ein AUTO_INCREMENT-Feld ist
newRow.Bestellnummer = autoIncrement;
mySqlCnnKrypton.Close();
```

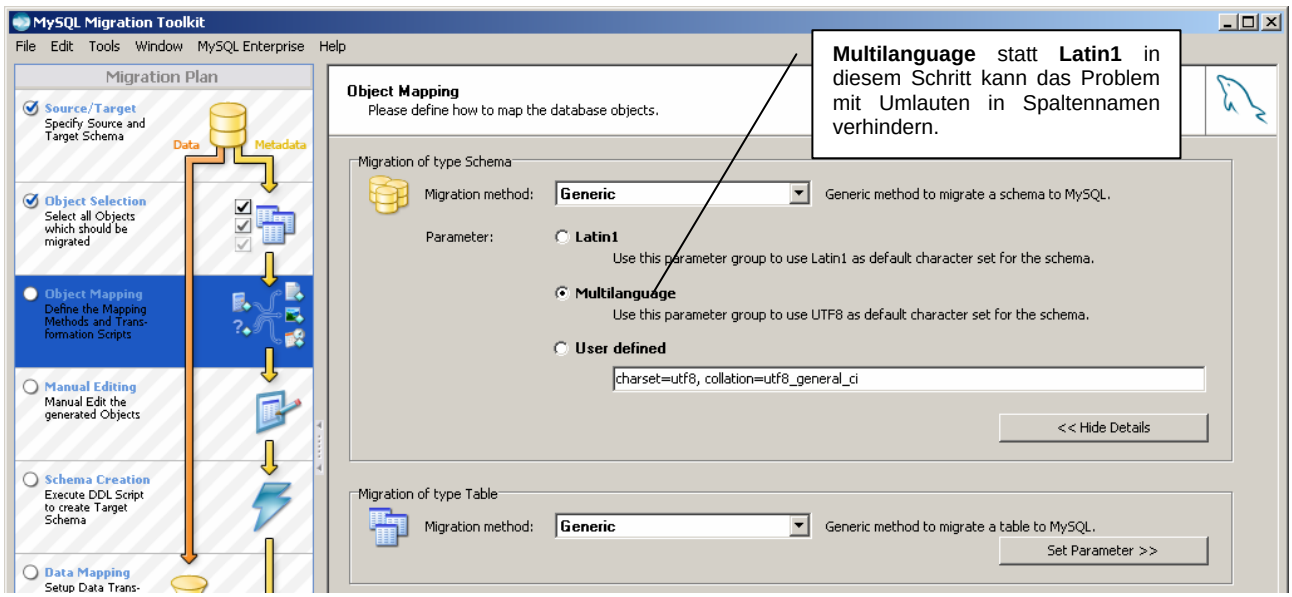
7.17.3 Probleme und Lösungen beim MySQL Zugriff

7.17.3.1 Connection-String ändert sich wenn Verbindung geöffnet wurde

Wurde eine MySqlConnection einmal geöffnet, verschwinden `User Id` und `Pwd` aus dem ConnectionString. Mit dem neuen ConnectionString ist es nicht möglich, eine neue Connection zu instanziieren und zu öffnen. Dies kann verhindert werden, indem man `Persist Security Info=true` in den ConnectionString einbaut. Allerdings wird sehr empfohlen dies nicht zu tun, da es zu Sicherheitsproblemen führt.

7.17.3.2 Probleme mit Umlauten in Feldnamen

Gibt es in einer MySQL-Tabelle Spalten in deren Namen Umlaute vorkommen, ist beim Lesen der Wert dieser Spalte jeweils DbNull.
Dieses Problem kann bei Migrieren von Access mit dem MySQL Migration Toolkit in manchen Fällen wie folgt verhindert werden:



Eine weitere Lösung ist das Verwenden einer View für das Lesen der Daten. In dieser View verwendet man einen Alias für die Spaltennamen mit Umlauten.

7.17.3.3 MySqlConnection erlaubt nur einen offenen DataReader

Im Gegensatz zur OleDbConnection, kann eine MySqlConnection nur einen geöffneten DataReader bedienen.

Lösungen:

1. Für jeden DataReader eine eigene Connection instanziieren.
2. Ergebnismenge in einer (untypisierten) DataTable zwischenspeichern, dies hat auch den Vorteil, dass man sich in der Ergebnismenge vorwärts und rückwärts bewegen kann.

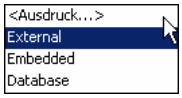
Beispiel für Lösung 2:

```
MySqlConnection mySqlCnn = new MySqlConnection();
mySqlCnn.ConnectionString = "Data Source=localhost; Database=krypton; …";
MySqlCommand cmdArtikel = mySqlCnn.CreateCommand();
cmdArtikel.CommandText = "SELECT * FROM qry_Artikel WHERE Artikelnummer = '30.10410'";
mySqlCnn.Open();
MySqlDataReader drArtikel = cmdArtikel.ExecuteReader();
DataTable dtArtikel = new DataTable();
dtArtikel.Load(drArtikel);
drArtikel.Close();
mySqlCnn.Close();
```

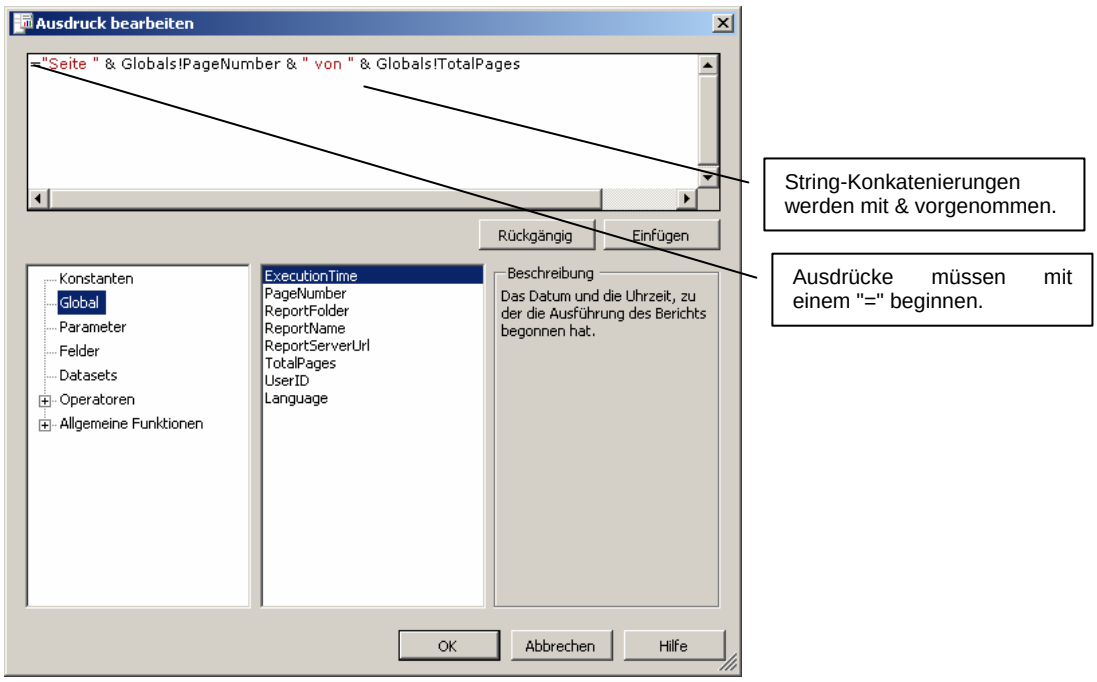
8 Microsoft Reporting Services

8.1 Ausdrücke

Diverse Eigenschaften von Steuerelementen haben im Eigenschaftenfenster eine ComboBox <Audruck…> als oberstem Eintrag.



Es öffnet sich der Dialog "Ausdruck bearbeiten".



8.1.1 Programmfluss mit IIf() steuern

Die Funktion `IIf()` verlangt 3 Parameter. Der erste ist ein Bool'scher Ausdruck, der zweite der Wert welcher zurückgegeben wird wenn der Bool'sche Ausdruck true ist, der dritte der Wert für den Fall dass der Bool'sche Ausdruck falsch ist.

8.1.2 Programmfluss mit Choose() steuern

`Choose()` übernimmt als ersten Parameter einen mit 1 beginnenden Index und als weitere Parameter einen "Parameterarray", d.h. mehrere weitere Parameter, die man zusammen als den Array anschauen kann (analog zu Parameterarrays in C#).
Der erste Parameter bestimmt, welches Element des Arrays zurückgegeben wird.

Beispiel:

```
= Choose(2, "file:/C:/test/Test1.jpg", "file:/C:/test/Test2.jpg")
```

Der Rückgabewert dieses Ausdrucks ist "file:/C:/test/Test2.jpg".

8.1.3 Programmfluss mit Switch() steuern

Die Funktion `Switch()` übernimmt immer eine gerade Anzahl Parameter, und zwar abwechslungsweise einen Bool'schen Ausdruck und einen Rückgabewert. Zurückgegeben wird jener Rückgabewert der hinter dem ersten Bool'schen Ausdruck kommt der sich zu true evaluiert.

8.1.4 Count, CountRows und CountDistinct

Mit `Count()` ist es nicht möglich, die Anzahl aller Datensätze zu ermitteln, sondern nur die Anzahl der Datensätze, welche in einem bestimmten Feld nicht den Wert DBNULL haben. Will man die Anzahl aller Datensätze ermitteln, muss man dazu die Funktion `CountRows("<DataSetName_Tabellenname>")` nehmen.

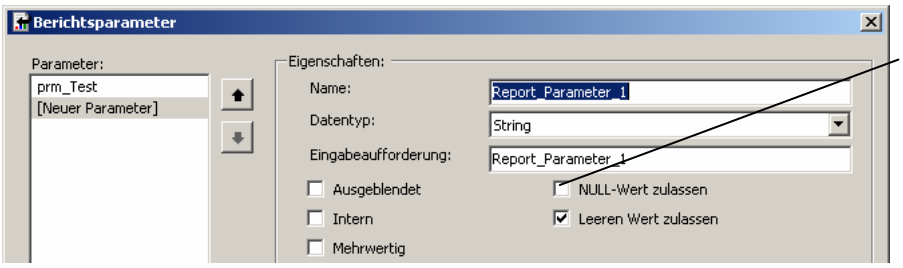
8.1.5 Prüfen ob ein Datenbankwert DbNull ist

Beispiel:

```
= IIf(IsNothing(First(Fields!Kommission.Value, "BelegpositionenCtrlDS_tbl_Belegkopf")), True, False)
```

8.2 Parameter

8.2.1 Allgemeines zu Parametern



Parameter mit aktiviertem "NULL-Wert zulassen" sind optional und müssen nicht übergeben werden.

Prüfen ob ein Parameter übergeben wurde:
`=IsNothing(Parameters!<Parametername>.Value)`

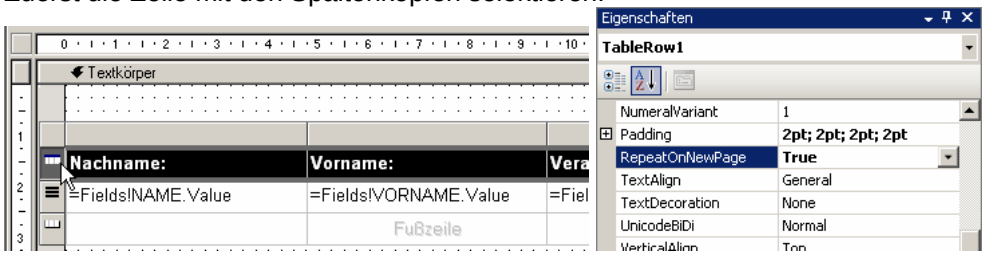
8.2.2 Parameter an Bericht übergeben

Die Methode `ReportViewer.LocalReport.SetParameters()` erwartet einen Array oder eine Collection vom Typ `Microsoft.Reporting.WinForms.ReportParameter`. Der Konstruktor von ReportParameter, erwartet einen Namen und einen Wert.

8.3 Tabellen

8.3.1 Spaltenköpfe auf allen Seiten wiederholen

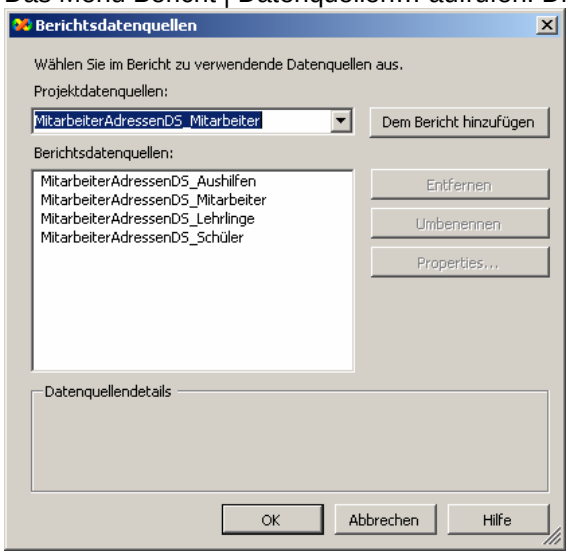
Zuerst die Zeile mit den Spaltenköpfen selektieren:



dann im Eigenschaftenfenster "RepeatOnNewPage" auf `true` setzen.

8.3.2 Mehrere Tabellen in einem Report

Das Menü Bericht | Datenquellen… aufrufen. Die benötigten Tabellen dem Bericht hinzufügen.

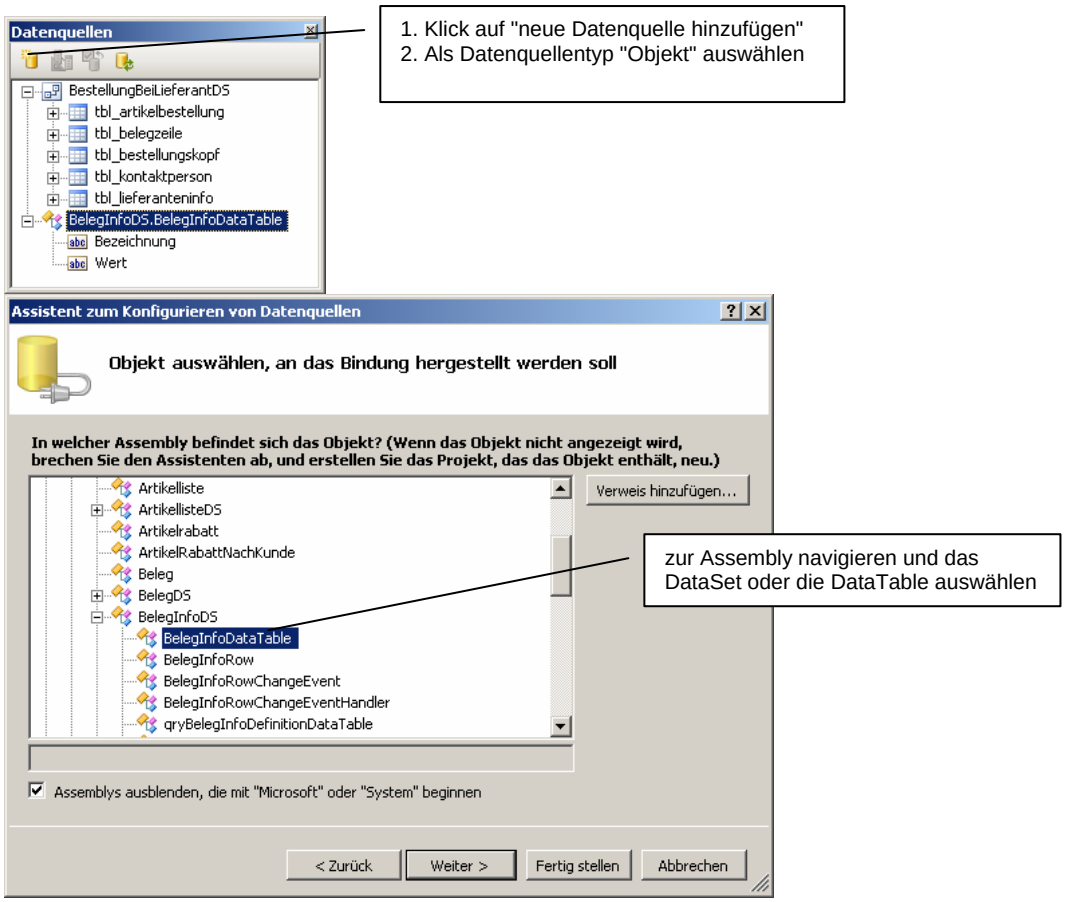


Im Bericht eine Tabelle selektieren. Im Eigenschaftenfenster die Eigenschaft `DataSetName` auf den Namen der gewünschten Berichtsdatenquelle setzen.

8.4 Datenquellen

8.4.1 Datenquellen-Fenster

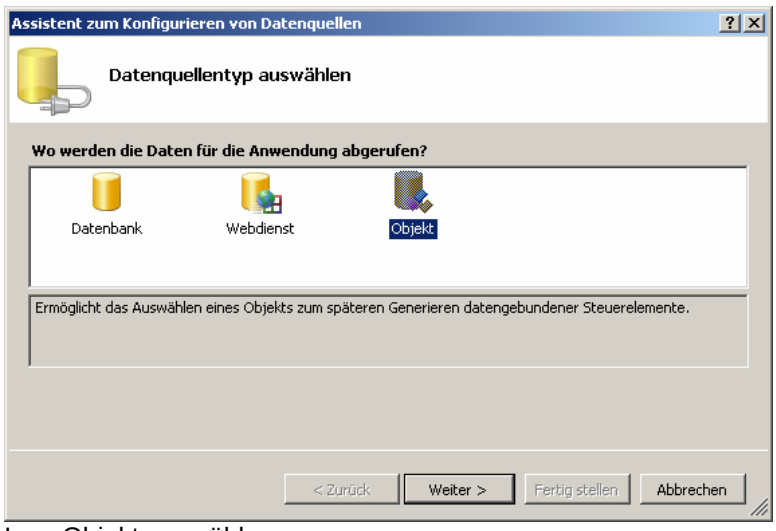
DataSet aus externer Assembly ins Datenquellen-Fenster aufnehmen:
Dies wird notwendig, wenn einem Report ein DataSet hinzugefügt werden soll, welches aus einer anderen Assembly kommt.



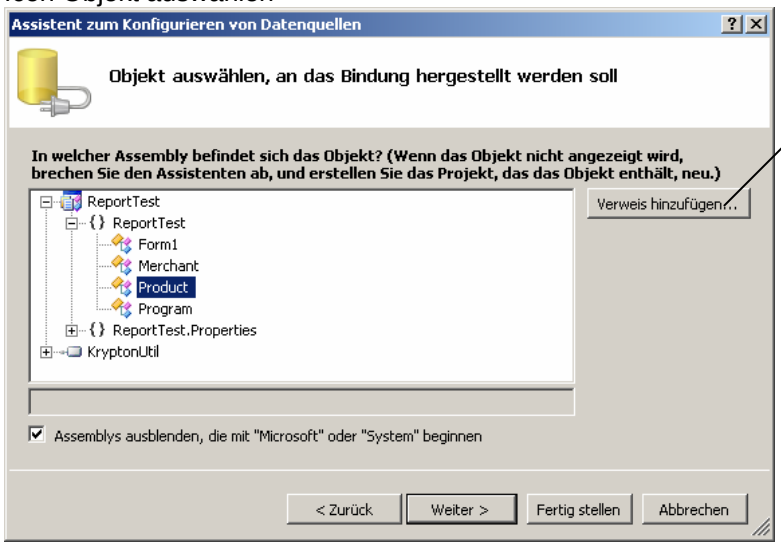
8.4.2 Objekt-Datenquellen

Vorgehensweise:

1. Eine "BusinessObject"-Klasse wie Product schreiben (darzustellende Werte als öffentliche Properties).
2. Eine Klasse "ProductCollection" schreiben, mit einer Methode, welche eine typisierte Liste z.B. System.Collections.Generic.List<Product> zurückgibt.
3. Eine BindingSource mit DataSource = typeof(Product) und DataSource = productCollection.GetProducts().
4. Eine ReportDataSource (Namespace: `Microsoft.Reporting.WinForms`) deren Name Eigenschaft <Namespace>_<BusinessObject> ist und deren Value Eigenschaft auf die BindingSource gesetzt wurde.
5. Im Report das Menü Daten | Neue Datenquelle hinzufügen… wählen.



Icon Objekt auswählen



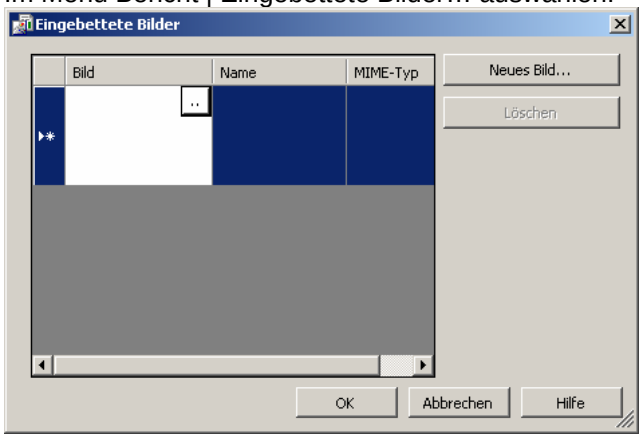
Man kann auch Business Klassen aus anderen Asemblies verwenden

Die Business-Klasse auswählen.
Die Business Klasse und ihre Eigenschaften erscheinen im Datenquellenfenster des Reports.

8.5 Grafiken

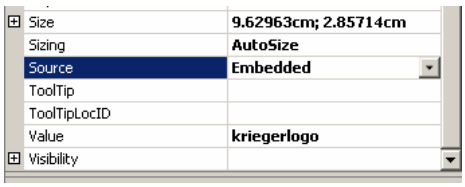
8.5.1 Eingebettete Bilder

Im Menü Bericht | Eingebettete Bilder… auswählen.



Mit Klick auf den Button "Neues Bild…" den Dateiauswahldialog öffnen und eine Grafik-Datei wählen

Aus der Toolbox ein Bild-Steuerelement auf den Report ziehen. Im Eigenschaftenfenster des Bild-Steuerelements die Eigenschaft `Source` auf `Embedded` setzen und die Eigenschaft `Value` auf das eingebettete Bild.



8.5.2 Externe Bilder

Im Bild-Steuerelement die Eigenschaft Source auf External setzen.
In der Eigenschaft Value den Pfag als URL angeben z.B: file:/C:/test/Test1.jpg.

8.6 ReportViewer

8.6.1 Verknüpfung mit ReportViewer

Arbeitet man im Projekt mit nur einem Report, lässt sich der Report grafisch mit dem ReportViewer verknüpfen.

Mit `reportViewer1.SetDisplayMode(DisplayMode.PrintLayout)` lässt sich der ReportViewer direkt in der Seitenlayout-Ansicht starten.

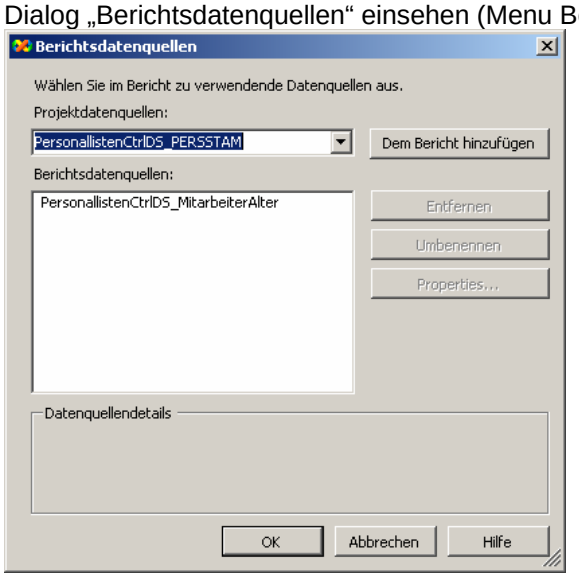
8.6.2 Mehrere Reports, ein ReportViewer

Definiert man im Projekt mehrere Reports, arbeitet jedoch mit nur einem ReportViewer, muss dem im Code dem ReportViewer den richtigen Report zuweisen.

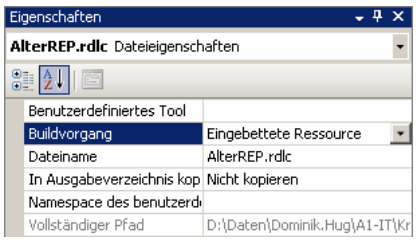
Beispiel:

```
// ReportDataSource instanziieren, das Argument besteht aus DataSet-Klassenname und Tabellenname
ReportDataSource rdsAlter = new ReportDataSource("PersonallistenCtrlDS_MitarbeiterAlter");
// Der ReportDataSource eine Instanz der Tabelle zuweisen
rdsAlter.Value = dsPersonallistenCtrl.MitarbeiterAlter;
// dem ReportViewer einen Report zuweisen
reportViewer.LocalReport.ReportEmbeddedResource = "Krypton.Net.AlterREP.rdlc";
// Die Datenquelle dem Report des ReportViewers zuweisen
reportViewer.LocalReport.DataSources.Add(rdsAlter);

reportViewer.RefreshReport();
```

Die für den Konstruktor von `ReportDataSource` zur Verfügung stehenden Datenquellen kann man im Dialog „Berichtsdatenquellen“ einsehen (Menu Bericht | Datenquellen…).



Für `reportViewer.LocalReport.ReportEmbeddetResource` muss der Namespace des Projekts gefolgt vom Dateinamen der rdlc-Datei angegeben werden. Wichtig ist, dass im Report als Buildvorgang „Eingebettete Ressource“ eingestellt ist.



8.6.3 ReportViewer konfigurieren

Beispiel:

```
frm.reportViewer1.SetDisplayMode(DisplayMode.PrintLayout);
frm.reportViewer1.ZoomMode = ZoomMode.Percent;
frm.reportViewer1.ZoomPercent = 100;
```

Sorgt dafür, dass der ReportViewer den Report im Seitenlayout mit Zoomfaktor 100% anzeigt.

9 Entwurfsmuster

9.1 Eine Factory als Singleton implementieren

Problemstellung: Controls und Business Classes benötigen eine Factory um an ihre Adapter zu kommen. Die Factory wird bereits im Konstruktor dieser Klassen benötigt. Ich will aber nicht jedem Objekt im Konstruktor einen Parameter Factory geben. Business Classes sind von Component abgeleitet damit man sie im Form Designer bentuzen und in die Toolbox platzieren kann. Components werden vom Form Designer in InitializeComponent() mit dem Konstruktor Component(System.ComponentModel.Component container) instanziiert und dies soll auch so bleiben.

Lösung:

Singletons sind Objekte die nur einmal existieren können. Sie haben einen privaten Konstruktor der nur von einem statischen Property aufgerufen wird. Das statische Property ist vom selben Typ wie das Singleton. Im Getter wird geprüft, ob schon eine Instanz existiert, wenn ja wird diese zurückgeben, wenn nein eine Instanz erzeugt, als private Variable gespeichert (für den nächsten Aufruf des Properties) und zurückgegeben.
Da das Property statisch ist, kann es überall mit dem Klassennamen ansprechen.

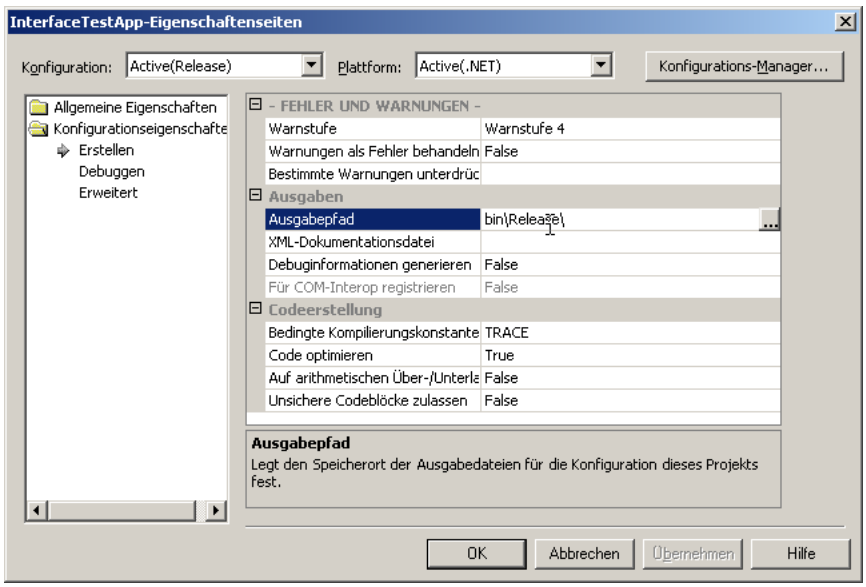
Beispiel für ein Singleton

```
public class Singleton {
   // die einzige Instanz die existieren kann
   private static Singleton myInstance;
   // Dieses Property ermöglicht Zugriff auf die Instanz
   public static Singleton Instance {
      get {
         // Falls keine Instanz vorhanden: erzeugen und speichern
         if (myInstance == null) myInstance = new Singleton();
         // Einzige Instanz zurückgeben
         return myInstance;
      }
   }
   // der private Konstruktor
   private Singleton() {}
}
// Jedes Objekt kann sich so eine Referenz auf das Singleton beschaffen
public class Client {
   . . .
   Singleton singleton1 = Singleton.Instance;
}
```

10 Visual Studio .NET 2003 IDE

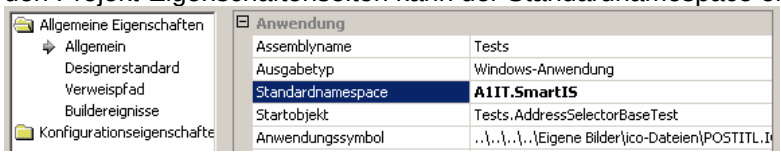
10.1 Ausgabepfad

Unter Projekt | Eigenschaften kommt man zu den Eigenschaftenseiten.
Unter Konfigurationseigenschaften | Erstellen (in der TreeView) kann man den Ausgabepfad festlegen.



10.2 Standardnamespace

Visual Studio platziert Klassen standardmässig in einem Namespace der gleich heisst wie das Projekt. In den Projekt-Eigenschaftenseiten kann der Standardnamespace eingestellt werden.



10.3 Versioning

Jede Assembly hat eine Version, welche in AssemblyInfo.cs resp. AssemblyInfo.vb im Attribut `[assembly: AssemblyVersion("1.0.3.9")]` festgelegt werden kann. Die Bedeutungen der vier Zahlen sind: Major, Minor, Build, Revision.
Standardmässig ist die auf `[assembly: AssemblyVersion("1.0.*")]` festgelegt. Dies bedeutet dass VS .NET bei der Kompilation für Build einen Wert generiert welcher der Anzahl Tage seit dem 1. Januar 2000 entspricht. Für Revision wird die Anzahl Sekunden die seit Mitternacht vergangen sind durch zwei geteilt. Beide Werte werden mit der Lokalzeit ermittelt. Legt man jedoch wie im oberen Beispiel alle vier Werte explizit fest, werden diese verwendet.
Zur Laufzeit kann die Version der verwendeten Assembly wie folgt ermittelt werden:

```
using System.Reflection;
...
Assembly assembly = Assembly.GetExecutingAssembly();
Version version = assembly.GetName().Version;
MessageBox.Show(version.ToString());
```

10.4 Klassen verschieben

Befinden sich beide Projekte in der gleichen Projektmappe, lässt sich dies mit Paste & Copy im Projektmappen-Explorer erledigen.
Befinden sich die Projekte in verschiedenen Projektmappen, sollte man temporär das Quellprojekt in die Projektmappe des Zielprojekt aufnehmen, dann ebenfalls im Projektmappen-Explorer mit Paste & Copy arbeiten.

11 VS .NET 2005: Änderungen

11.1 Formulare

Der Code für Formulare befindet sich neuerdings in zwei Dateien. Z.B. Form1.cs und Form1.Designer.cs. In Form1.cs befindet sich der selbst geschriebene Code. In Form1.Designer.cs die Region "Vom Windows Form-Designer generierter Code" sowie die Methode Dispose und die Deklaration von Controls und Komponenten.

11.2 Steuerelemente

11.2.1 MenuStrip

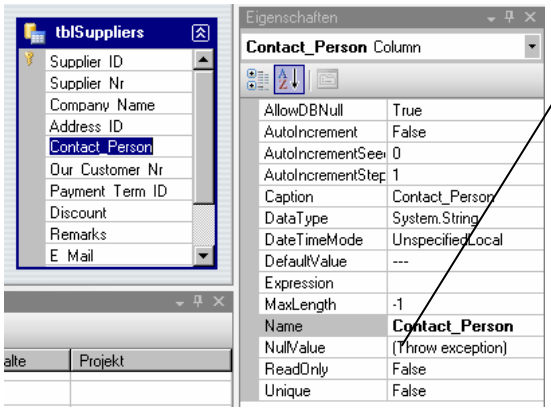
Ersetzt MainMenu. Mit MenuStrip ist es mit Leichtigkeit möglich, zu einem Menüeintrag auch ein Icon darzustellen (Eigenschaft Image).

11.3 Datenzugriff

Erzeugt man ein typisiertes DataSet, wird für jede Tabelle automatisch ein "typisierter" TableAdapter erzeugt. Diese Klassen erscheinen automatisch in der ToolBox ganz oben.

11.3.1 DataSets

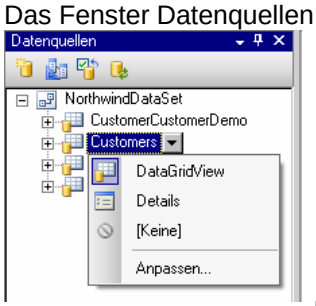
Bindet man eine DataTable an TextBoxen kommt es zu Problemen wenn der Datensatz DBNull-Werte enthält. Im XML-Designer kann man für die Felder einer DataTable Werte angeben. Diese werden anstelle von DBNull verwendet.



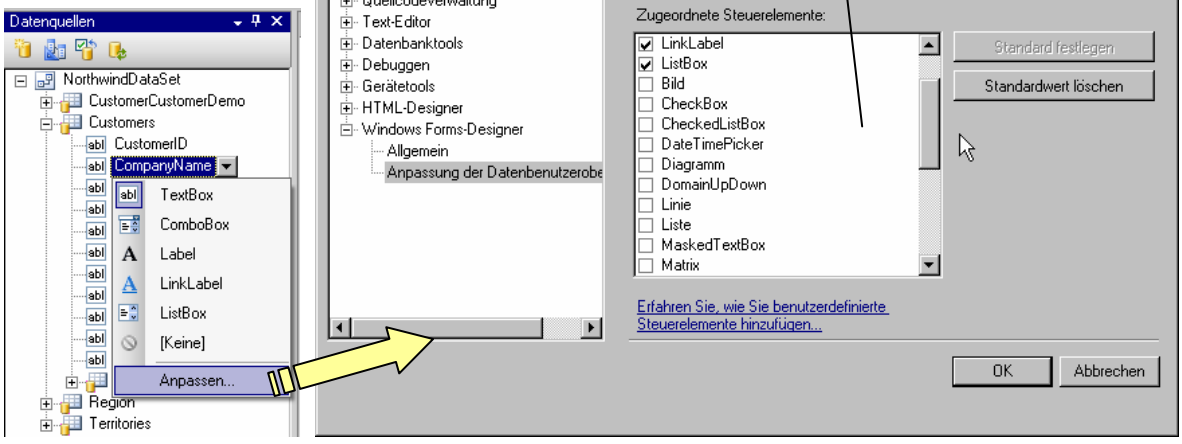
In einer DataTable hat jede Column eine Eigenschaft NullValue, mit welcher man den zurückzugebenden Wert festlegen kann, für den Fall dass der Wert DBNull ist.

11.3.2 Datenquellenfenster

Das Fenster Datenquellen erscheint standardmässig am gleichen Ort wie die ToolBox.

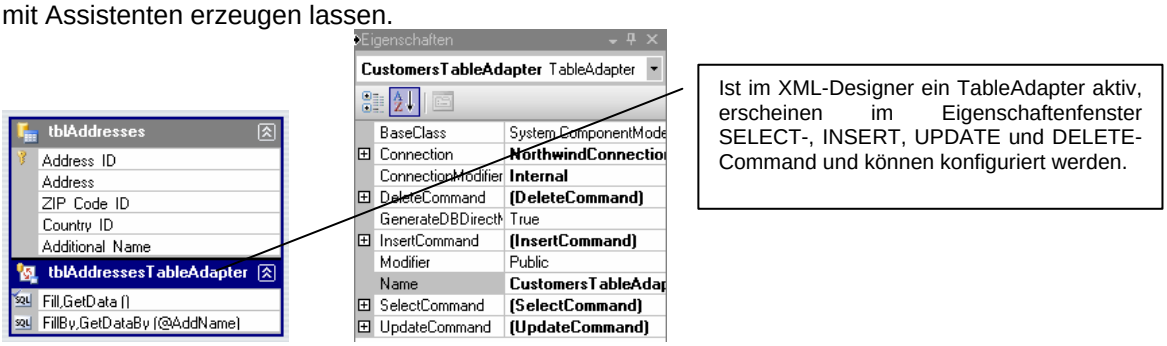


Im Datenquellenfenster können weitere Steuerelemente für die "Drag & Drop"-Bindung verfügbar gemacht werden.



11.3.3 TableAdapter

Der TableAdapter ist das typisierte Äquivalent zum DataAdapter. Er lässt sich ähnlich wie ein DataAdapter benutzen. Der TableAdapter kann über mehrere Fill() Methoden verfügen, man kann zusätzliche Fill() Methoden (z.B. solche die Parameterwerte für die WHERE-Klausel übernehmen) bequem im XML Designer mit Assistenten erzeugen lassen.



Ist im XML-Designer ein TableAdapter aktiv, erscheinen im Eigenschaftenfenster SELECT-, INSERT, UPDATE und DELETE-Command und können konfiguriert werden.

Der TableAdapter verfügt über eine Connection zur DB. Man kann spezifizieren, dass der ConnectionString automatisch im Applikationskonfigurationsfile gespeichert wird.
Eine Klasse TableAdapter oder SqlTableAdapter usw. gibt es nicht. Beim Erzeugen eines typisierten DataSets wird für jede Tabelle ein TableAdapter erzeugt. Diese sind jeweils von System.ComponentModel.Component abgeleitet und verfügt über eine Connection, einen (noch alten) DataAdapter und Commands. Diese sind jeweils typisiert (also SqlConnection oder OleDbConnection usw.)
Um Datenzugriffsmethoden (Fill(), Update(), InsertQuery(), UpdateQuery() usw.) aufzurufen muss man die Connection nicht selbst öffnen. Ist die Connection vor dem Aufruf geschlossen, öffnet sie der TableAdapter selbst und schliesst sie danach wieder. Ist sie vor dem Aufruf geöffnet, lässt der TableAdapter sie offen.
Tipp!: Macht eine Methode mehrere Datenzugriffe mit TableAdaptern, so lohnt es sich zu Beginn der Methode die Connection zu öffnen und erst am Schluss wieder zu schliessen (Bei KundenCtrl konnten so ca. 1.7 Sekunden gespart werden).

Vorsicht bei Parametern vom Typ char und nchar!
Parameter vom Typ char und nchar haben standardmässig eine Länge (Size) die der Feldlänge in der Datenbank entspricht. Dies bedeutet, dass der Wert automatisch mit Leerzeichen aufgefüllt wird bis die Länge erreicht ist.
Bei Parametern für Abfragen mit dem Vergleichsoperator "=" in der WHERE Klausel macht dies Sinn, da SQL Server bei diesen beiden Datentypen in den Tabellen die Werte auch automatisch mit Leerzeichen auffüllt.
Verwendet man aber den Parameter mit dem Operator "LIKE" und Wildcards wird dies ein störender Nebeneffekt, da z.B. aus "A%" "A% " wird (bei einer Länge von 10 werden also noch 8 Leerzeichen angehängt). Dies verfälscht die Ergebnismenge.
Lösung: Für die Eigenschaft Size des Parameters den Wert -1 eingeben (dieser wird zwar automatisch auf 2147483647 geändert, das Problem ist aber gelöst).

Insert-, Update- und Delete-Methoden

Der automatisch generierte Code des TableAdapters:

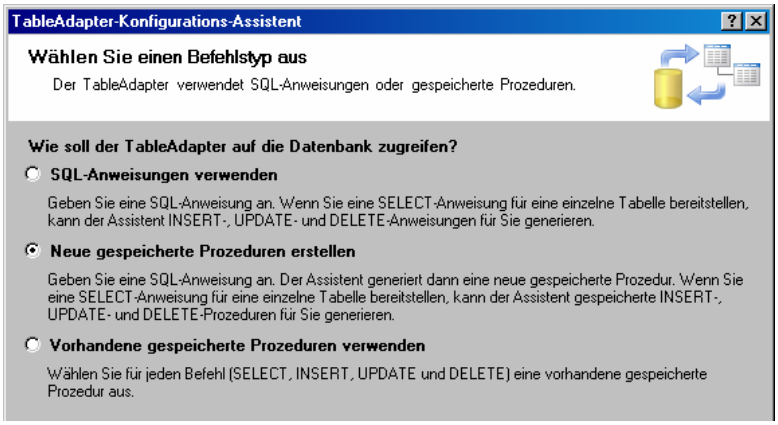
```
Public Overloads Overridable Function InsertQuery(ByVal Datum As System.Nullable(Of Date), By
    Dim command As System.Data.OleDb.OleDbCommand = Me.CommandCollection(2)
    If (Datum.HasValue = true) Then
        command.Parameters(0).Value = CType(Datum.Value,Date)
    Else
        command.Parameters(0).Value = System.DBNull.Value
    End If
```

Ist ein Argument null, so wird dem Datenbankparameter automatisch der Wert DBNull.Value zugewiesen

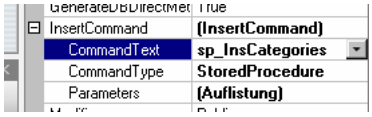
Hier werden **generische Typen** – nämlich System.Nullable(Of Date) – verwendet.

11.3.4 TableAdapter für Verwendung von Stored Procedures konfigurieren

In VS .NET 2003 konnte man einen DataAdapter aufs Formular ziehen und mit Klick auf den Hyperlink "DataAdapter konfigurieren" u.a. auch einstellen, dass der für den Datenzugriff vorhandene Stored Procedures verwendet, resp. dafür in der Datenbank neue erzeugt.
In VS 2005 muss man anders vorgehen. Statt im XML DataSet-Designer aus dem Server Explorer eine Tabelle hineinzuziehen, nimmt man einen TableAdapter aus der Toolbox. Es erscheint der TableAdapter-Konfigurations-Assistent.
Auf der ersten Seite kann man die Connection zur Datenbank wählen.
Die zweite Seite ermöglicht, für den Datenzugriff statt SQL-Anweisungen Stored Procedures zu verwenden oder dafür neue zu erzeugen.

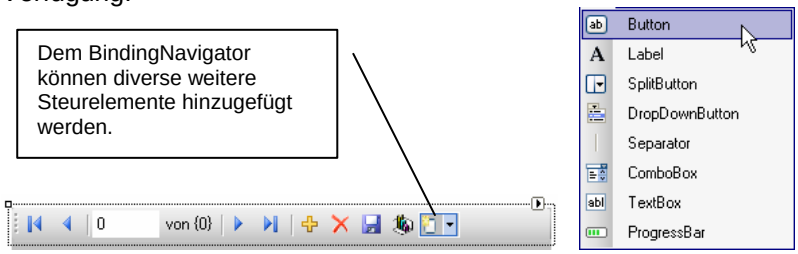


Es ist auch möglich, einen existierenden TableAdapter dazu zu bringen mit Stored Procedures zu arbeiten. Dazu klickt man ihn im XML-DataSet-Designer an und kann danach im Eigenschaftenfenster ein Command auswählen. Setzt man CommandType auf StoredProcedure kann unter CommandText eine vorhandene Stored Procedure auswählen. Die Parameter Auflistung wird danach automatisch angepasst.



11.3.5 BindingNavigator

Namespace: System.Windows.Forms. Stellt eine Leiste zur Navigation zwischen den Datensätzen zur Verfügung.



11.3.6 BindingSource

Ist eine Schicht zwischen der Datenquelle (DataTable o.ä.) und dem Control. Dies ermöglicht, mehrere Controls an die gleiche Datenquelle zu binden und trotzdem für jede Bindung einen eigenen aktuellen Datensatz zu haben.

Via BindingSource einen Wert aus der gebundenen Tabelle lesen:

```
string artikelNr = ((DataRowView)bsArtikelSuchen.Current)["Artikelnummer"].ToString();
```

VORSICHT:

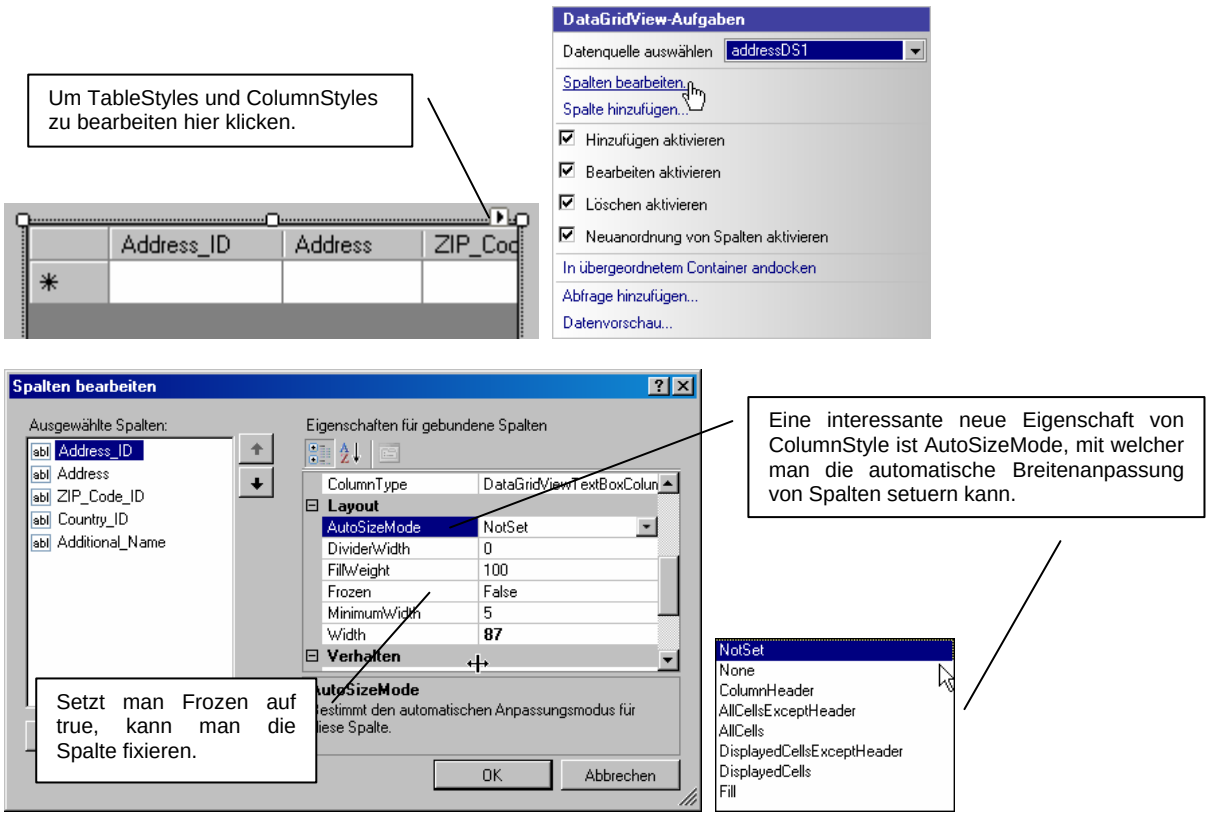
Das Ereignis `BindingSource.PositionChanged` wird auch ausgelöst wenn man mit `TableAdapter.Fill()` die an die BindingSource gebundene Tabelle neu füllt (falls die Eigenschaft `ClearBeforeFill` auf `true` gesetzt ist).

11.4 DataGridView

Die Eigenschaften DataSource und DataMember können wie beim DataGrid verwendet werden.

11.4.1 DataGridViewTextBoxColumn

DataGridTableStyles und DataGridColumnStyles können jetzt im Eigenschaftenfenster unter Columns bearbeitet werden. Eine weitere Möglichkeit ist:



Die Zellen einer Spalte können neu auch durch ComboBoxes, Buttons oder Images dargestellt werden.

11.4.2 DataGridViewButtonColumn

Damit der Wert des Text Properties von DataGridViewButtonColumn in den Buttons auch angezeigt wird, muss man `DataGridViewButtonColumn.UseColumnTextForButtonValue` auf `true` setzen.

11.4.3 Wichtige Eigenschaften von DataGridView

`bool` **`AllowUserToOrderColumns`**
Legt fest, ob der User Reihenfolge der Spalte ändern kann.

`SelectionMode`
Setzt man diese Eigenschaft auf `FullRowSelect`, wird immer eine ganze Zeile selektiert.

`AutoSizeColumnsMode`
Legt AutSizeMode gleich für alle Spalten in der DataGridView fest.

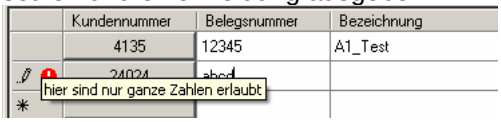
`public DataGridViewCell` **`CurrentCell`** `{ get, set }`
Die im Moment bearbeitete Zelle.

11.4.4 Datenformatierung

Das Ereignis `CellFormatting` wird immer ausgelöst, wenn Daten in einer Zelle formatiert werden müssen. Über die Eigenschaften `ColumnIndex`, `RowIndex`, `DesiredType` und `Value` der EventArgs kann man feststellen, welche Daten in welcher Zelle formatiert werden müssen.

11.4.5 Validierung

Das Ereignis `DataGridView.DataError` abfangen. Im Eventhandler herausfinden, welche Zelle welches Problem hat. Mit `DataGridView.CurrentRow.ErrorText` kann man einen ErrorProvider auf die Zeile setzen und eine Meldung ausgeben.

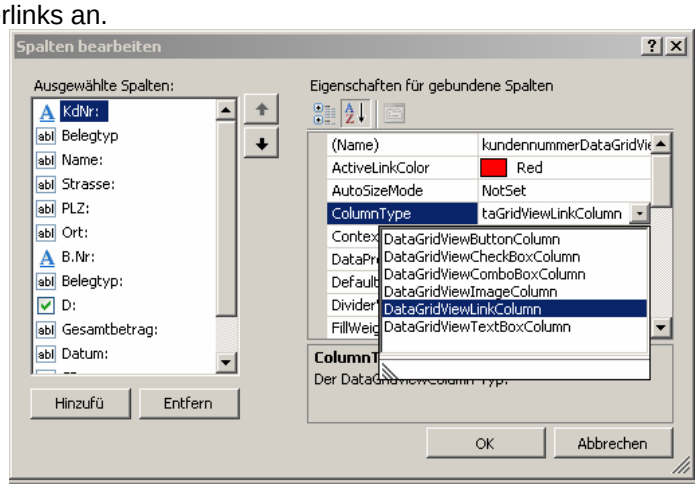


Danach muss man aber auch dafür sorgen, dass der ErrorProvider wieder verschwindet, falls die Falscheingabe korrigiert wurde. Dafür gilt es, das `CellValidated` Ereignis abzufangen. Da der User eine

Zelle mit ungültigen Daten nicht verlassen kann, handelt es sich beim nächsten `CellValidated` Ereignis um die gleiche Zelle. Im Eventhandler kann man mit `DataGridView.IsCurrentCellDirty` abfragen, ob die Zelle nun einen gültigen Wert enthält und mit `DataGridView.CurrentRow.ErrorText = ""` den ErrorProvider wieder entfernen.

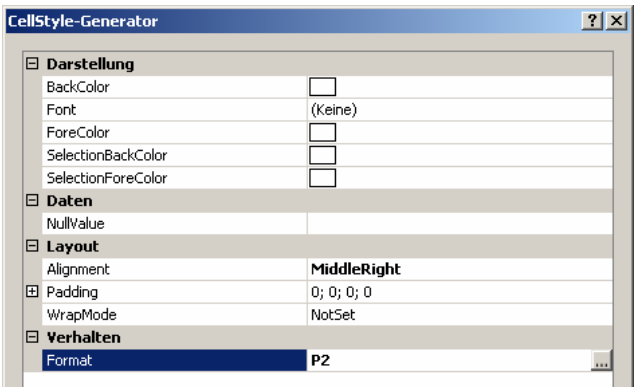
11.4.6 DataGridViewLinkColumn

Zeigt die Werte als Hyperlinks an.



Um auf das Klicken auf einen Hyperlink zu reagieren muss man das CellContentClick-Event der DataGridView behandeln.

11.4.7 Arbeiten mit Prozentwerten



Die Format-Eigenschaft im CellStyle-Generator auf P*N* setzen (N sind die Anzahl Nachkommastellen). Dies bewirkt, dass `0.1` als `10%` dargestellt wird.

```
/// <summary>Verhindert Eingaben die grösser als 100 und kleiner als 0 sind.</summary>
private void dgvRabatte_CellValidating(object sender, DataGridViewCellValidatingEventArgs e) {
    if (dgvRabatte.Columns[e.ColumnIndex].Name == "clmRabatt") {
        string strVal = e.FormattedValue.ToString();
        // notwendig, da das Validating-Event auch ausgelöst wird,
        // wenn der user mit den Pfeiltasten über die Zellen navigiert
        strVal = strVal.Replace("%", "");
        double dblVal = 0;
        try {
            dblVal = Convert.ToDouble(strVal);
            dgvRabatte.Rows[e.RowIndex].ErrorText = "";
        }
        catch (FormatException) {
            dgvRabatte.Rows[e.RowIndex].ErrorText = "Geben Sie eine Zahl zwischen 0 und 100 ein";
            e.Cancel = true;
            return;
        }
        if (dblVal > 100 || dblVal < 0) {
            dgvRabatte.Rows[e.RowIndex].ErrorText = "Geben Sie eine Zahl zwischen 0 und 100 ein";
            e.Cancel = true;
        }
    }
}

/// <summary>Verhindert hässliche (automatische) MessageBox-Fehlermeldung. </summary>
private void dgvRabatte_DataError(object sender, DataGridViewDataErrorEventArgs e) {
    dgvRabatte.Rows[e.RowIndex].ErrorText = "Geben Sie eine Zahl von 0 - 100 ohne %-Zeichen ein";
```

```
    e.Cancel = true;
}

/// <summary>Dividiert den eingegebenen Wert durch 100,
/// damit z.B. 10% als 0.1 in DB gespeichert wird.</summary>
private void dgvRabatte_CellEndEdit(object sender, DataGridViewCellEventArgs e) {
    if (dgvRabatte.Columns[e.ColumnIndex].Name == "clmRabatt") {
        double dblVal;
        try {
            dblVal = Convert.ToDouble(dgvRabatte.Rows[e.RowIndex].Cells[e.ColumnIndex].Value);
            dblVal = dblVal / 100;
            dgvRabatte.Rows[e.RowIndex].Cells[e.ColumnIndex].Value = dblVal;
            dgvRabatte.Rows[e.RowIndex].ErrorText = "";
        }
        catch (FormatException) {
            dgvRabatte.Rows[e.RowIndex].ErrorText = "Geben Sie eine Zahl zwischen 0 und 100 ein";
        }
    }
}
```

11.4.8 Wichtige Ereignisse von DataGridView

`CurrentCellDirtyStateChanged`
Tritt ein, wenn der User beginnt, den Wert in der aktuellen Zelle zu ändern.

`DataError`
Tritt ein, wenn ein Validierungsvorgang eine Ausnahme wirft und ermöglicht, die Ausnahme zu behandeln.
Mit `e.Exception` kann man auf die geworfene Exception zugreifen.
Mit `e.RowIndex` und `e.ColumnIndex` kann man die betreffende Zelle ermitteln.
Hat man den Fehler erfolgreich behoben, sollte man das Event mit e.Cancel = true; abbrechen.

Beispiel:

```
private void dgv_DataError(object sender, DataGridViewDataErrorEventArgs e) {
       string defaultValue = …;
       dgv.Rows[e.RowIndex].Cells[e.ColumnIndex].Value = defaultValue;
       e.Cancel = true;
}
```

`EditingControlShowing`
Wird ausgelöst, wenn das Control zum Bearbeiten des Zelleninhaltes angezeigt wird (meist eine TextBox).
Im EventHandler kann man mit `e.CellStyle` die Darstellung beeinflussen.
`e.CellStyle.BackColor = Color.Aquamarine;`

`UserDeletingRow`
Wird ausgelöst, wenn der User über die DELETE-Taste eine Zeile zu löschen versucht. Versucht der Benutzer über den BindingNavigator eine Zeile zu löschen, wird dieses Ereignis nicht ausgelöst.
`e.Row`: gibt die zu löschende Zeile als DataGridViewRow zurück.
`e.Cancel`: Indem man diese Eigenschaft auf true setzt, verhindert man das Löschen der Zeile.

11.4.9 Weiteres zu DataGridView Ereignissen

Klickt der User auf eine Zelle, werden die folgenden Events in dieser Reihenfolge ausgelöst:
`Click`
`CellClick`
`CellContentClick`

Springt der User mit den Pfeiltasten von einer Zelle zur anderen werden folgende Events in dieser Reihenfolge ausgelöst.
`CellLeave`
`CellValidating`
`CellValidated`
`CellStateChanged`
`CellStateChanged`
`CurrentCellChanged`
`CellEnter`
`SelectionChanged`
Beginnt er danach, den Wert über die Tastatur zu ändern, kommen folgenden Ereignisse hinzu:
`CellBeginEdit`

`ControlAdded`
`EditingControlShowing`
`CurrentCellDirtyStateChanged`

`CellLeave` wird jeweils vor `RowLeave` ausgelöst.
`CellValidating` / `CellValidated` werden jeweils vor `RowValidating` / `RowValidated` ausgelöst.

Verlässt der User eine Zelle die er bearbeitet hat, wird zuerst `CellValidating`, danach `CellEndEdit` ausgelöst.

11.4.10 Behandeln von Tastatur-Ereignissen

Das DataGridView scheint kein `KeyPress`-Event auszulösen. Mit dem Event `KeyDown` hingegen kann man alle Tasten (auch RETURN, Pfeiltasten usw. abfangen).

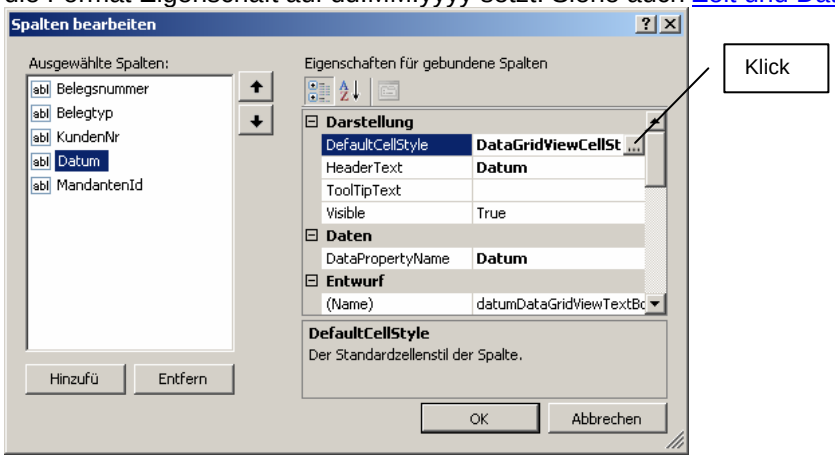
Beispiel:

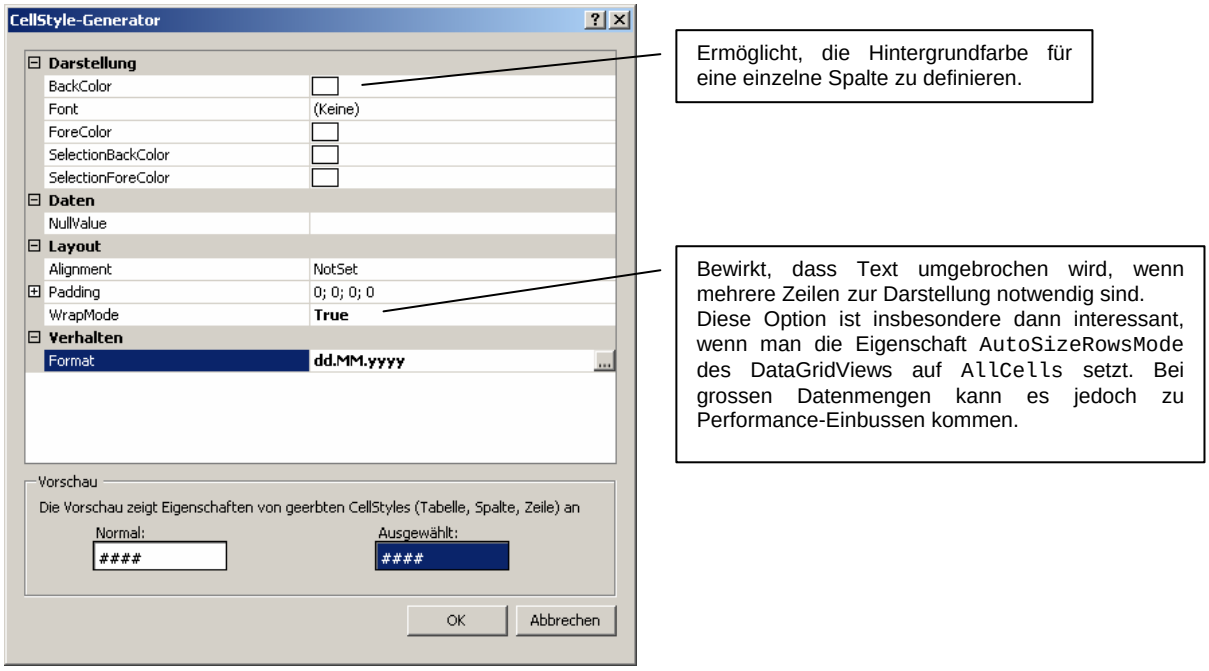
```
private void dataGridView1_KeyDown(object sender, KeyEventArgs e) {
    Debug.WriteLine(e.KeyCode.ToString());
    if ((e.KeyCode == Keys.Down || e.KeyCode == Keys.Return) &&
        dataGridView1.CurrentRow.Index == dataGridView1.Rows.Count - 1) {
        dataGridView1.Rows.Add();
    }
}
```

11.4.11 DataGridView: Tipps und Tricks

Wenn der User eine neue Zeile einfügt oder eine bestehende Zeile ändert und die Zeile hat den Fokus nie beim Update() mit dem TableAdapter die Fokus nie verloren, werden die Änderungen nicht erkannt und nicht gespeichert. Man kann das Problem lösen, indem man vor dem Aufruf von TableAdapter.Update() this.Validate() aufruft.
Alle Controls (also auch die Klasse Form und UserControl) verfügen über die Methode Validate(). In der .NET Framework 2.0 Dokumentation steht dazu folgendes:
"Überprüft den Wert des Steuerelements, das den Fokus verliert, indem das Validating-Ereignis und das Validated-Ereignis in dieser Reihenfolge ausgelöst werden."

Datums-Spalten können auf das Format 24.12.2007 formatiert werden indem man im DataGridViewCellStyle die Format Eigenschaft auf dd.MM.yyyy setzt. Siehe auch Zeit und Datum formatieren.





11.4.11.1 ToolTips

Damit ToolTips richtig angezeigt werden, muss `ShowCellToolTips` auf `false` setzen.

11.4.11.2 DbNull

Haben weder der User noch die Applikation einen Wert in eine Zelle geschrieben, so evaluiert sich `Cell.Value == System.DbNull.Value` zu `true`.

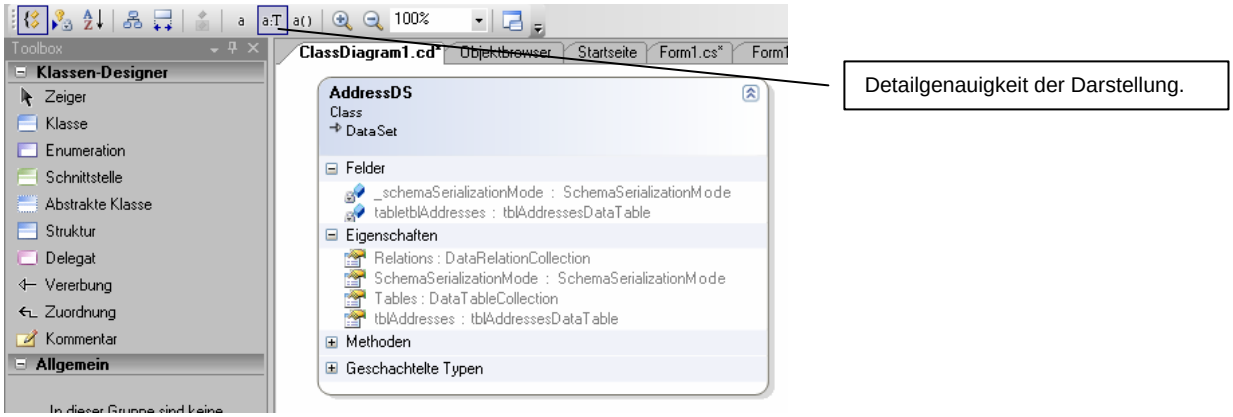
11.4.11.3 Anzeige aktualisieren

Verwendet man das DataGridView in Kombination mit einem Form und lässt den User die Daten nur über Controls des Forms ändern, aktualisiert sich die CurrentRow des DataGridViews nicht automatisch. Die Methode `DataGridView.InvalidateRow()` schafft abhilfe.

11.5 Wichtige Neuerungen der IDE

11.5.1 Klassendiagramme

Neuerdings kann man sich automatisch Klassendiagramme erzeugen lassen. Dazu muss man zuerst im Menü Ansicht die Klassenansicht einblenden. Anschliessend kann man in der Klassenansicht im Kontextmenü einer dort aufgeführten Klasse per Klick ein Klassendiagramm erzeugen.



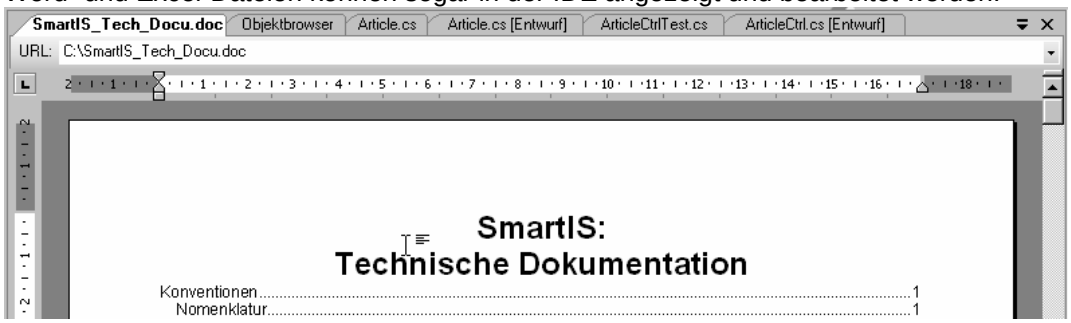
Im Kontextmenü eines Klassendiagramms kann man unter Hinzufügen einen neuen Member für die Klasse erzeugen. Dabei wird gleich automatisch das entsprechende Codegerüst erstellt.

11.5.2 Hyperlinks im Code

In Kommentaren kann man Hyperlinks auf Dateien, URL's oder E-Mail Adressen platzieren. Bei Hyperlinks auf Dateien (mit `file:///...`) darf der Pfad aber keine Leerzeichen enthalten.

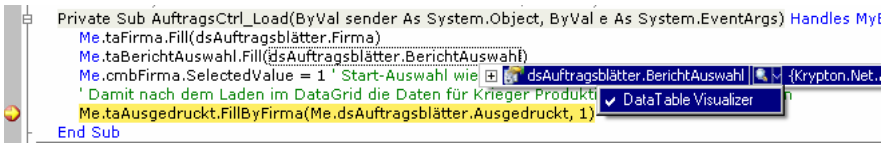
```
using System.Windows.Forms;

// file:///C:/SmartIS_Tech_Docu.doc

namespace A1IT.SmartIS {
```

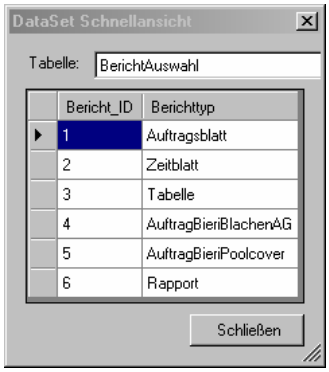
Word- und Excel-Dateien können sogar in der IDE angezeigt und bearbeitet werden.



12 Debugging

12.1 DataTable Visualizer

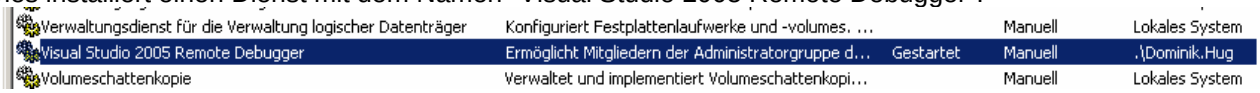




12.2 Remote Debugging

Damit Remote Debugging möglich wird, müssen auf der Remote Machine die Remote Components installiert werden. Diese findet man bei den Visual Studio 2005 Installationsdateien unter `Remote Debugger\X86\rdbgsetup.exe`.

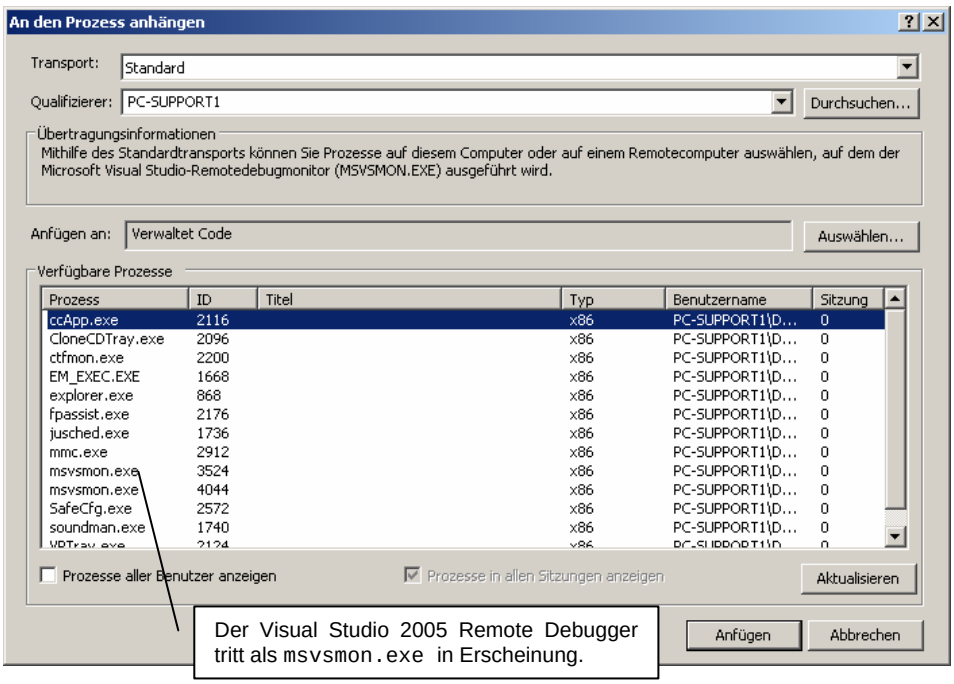
Dies installiert einen Dienst mit dem Namen "Visual Studio 2005 Remote Debugger".



Auf der Remote Machine muss ein Benutzerkonto mit gleichem Namen und Passwort wie das lokale Benutzerkonto existieren (dann funktioniert es im gleichen Netzwerk).

Die zu debuggende Applikation muss im DEBUG-Modus kompiliert sein und die PDB-Datei die beim Kompilieren erzeugt wird und unter \bin\Debug platziert wird muss ebenfalls auf der Remote Machine sein.

Läuft die Applikation auf der Remote Machine, startet man lokal VS 2005 und öffnet das Projekt der betreffenden Applikation.
Im Menü Debuggen wählt man den Eintrag "An den Prozess anhängen…"

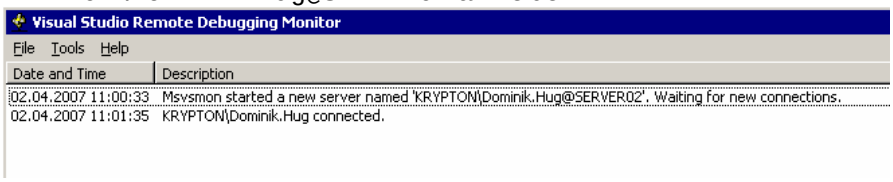


In der ComboBox "Transport:" wählt man "Standard".
In der ComboBox "Qualifizierer:" sucht man den Remote-PC.
Bei "Anfügen an:" ist "Verwaltet Code" (managed code) die richtige Auswahl.
Danach muss man nur noch unter "Verfügbare Prozesse" die richtige Applikation auswählen.
Hat man auf "Anfügen" geklickt, kann man Breakpoints setzen.
Wird auf der Remote Machine ein Breakpoint erreicht, stoppt der Prozess und man kann in Visual Studio debuggen (Variablen überwachen usw.).

12.2.1 Remote Debugging bei Krieger AG

Auf SERVER02 von Krieger AG brachte ich Remote Debugging wie folgt zum laufen:

1. `rdgbsetup.exe` ausführen.
2. Im Startmenü unter Programme\Microsoft Visual Studio 2005\Visual Studio Tools\Visual Studio 2005 Remote Debugger ausführen.
3. (lokal) in Visual Studio 2005 mit Debuggen\An Prozess anhängen… mich mit `KRYPTON\Dominik.Hug@SERVER02` anmelden.



Mit dieser Vorgehensweise klappte das Remote Debugging, unabhängig davon ob der Service "Visual Studio 2005 Remote Debugger" lief oder nicht.

12.3 Debugging Tipps und Tricks

12.3.1 Methoden beim Debugging überspringen

Das Attribut `[DebuggerStepThrough()]` lässt sich auf Klassen, Strukturen, Methoden und Konstruktoren anwenden. Es bewirkt, dass der Debugger im Einzelschrittmodus das entsprechende Element überspringt.

13 Performance

13.1 Performance Tests

13.1.1 StopWatch

Im Namespace `System.Diagnostics` befindet sich die Klasse `StopWatch`. Mit ihr kann man auf Millisekunden genau messen. Problematisch ist aber, dass jedes Objekt dessen Performance wir messen

wollen, eine Referenz auf die gleiche Instanz von `StopWatch` benötigt, was etwas kompliziert zu realisieren ist.

13.1.2 DateTime-Struktur

Eine interessante Alternative bietet die `DateTime`-Struktur. Die Eigenschaft `DateTime.Now.Ticks` gibt die Anzahl von 100 Nanosekunden Intervallen zurück, die seit dem 1. Januar 01, 0:00:00 vergangen sind (sprengt nicht den Wertebereich von Long, dessen Maximalwert liegt bei 9'223'372'036'854'775'807.) Mit DateTime haben wir also eine 10'000 Mal höhere zeitliche Auflösung als mit StopWatch.
Zusätzlich bietet `DateTime.Now.Ticks` die Möglichkeit, überall die Zeit verfügbar zu haben, welche seit dem Programmstart vergangen ist (mit `StopWatch` müssten wir eine Instanz davon in jedem Objekt verfügbar machen).

13.2 Performance von Structs

Dass Struktures Wertetypen sind kann sich positiv und negativ auf die Performance auswirken. Die Speicherallokation ist sehr schnell, da dies auf dem Stack geschieht. Dasselbe gilt für das Zerstören von Strukturen.
Übergibt man allerdings einen Struct als Parameter oder weist einem Struct einen anderen zu (A = B), werden die gasamten Daten der Struktur kopiert, während bei Klassen nur die Referenz kopiert wird. Der Performanceverlust hängt also stark von der Grösse der Struktur ab, was die Tatsache unterstreicht, dass Structs nur für kleine Datenstrukturen verwendet werden sollten. Um Strukturen mit hoher Performance einer Methode zu übergeben, kann man den Parameter mit dem Schlüsselwort `ref` übergeben. Dann wird nur dessen Speicheradresse kopiert, was gleich schnell geht wie eine Klasse zu übergeben.

13.3 Performance von Arrays, Collections usw.

Die Klasse `Dictionary` bietet eine ausgefeilte Datenstruktur welche Datenzugriff aufgrund eines Schlüssels von beliebigem Typ ermöglicht. Das Hinzufügen und Entfernen von Elementen geschieht mit höherer Performance als bei der `ArrayList`, da keine nachfolgenden Elemente verschoben werden müssen.

13.4 Performance: weiteres

Boxing und Unboxing hat negative Auswirkungen auf die Performance.

14 Windows Services

Die Klasse `ServiceController` im Namespace `System.ServiceProcess` und in der DLL System.ServiceProcess.dll repräsentiert einen Windows Service.
Die statische Methode `GetServices()` gibt einen Array von ServiceController Objekten mit allen installierten Services zurück. Die Überladung `GetServices(string machineName)` macht das gleiche für einen anderen Computer im Netzwerk.
Die Eigenschaft `ServiceName` von ServiceController identifiziert einen Service eindeutig.
Die wichtigsten Methoden sind `Start()`, `Stop()`, `Pause()` und `Continue()`.
Interessante Eigenschaften sind `CanPauseAndContinue`, `DependentServices`, `ServicesDependentOn` und `Status`.
Der Nachrichtendienst (welcher für net send... benötigt wird) hat den Namen "Messenger".

15 Asynchrone Methodenaufrufe

15.1 Asynchroner Methodenaufruf mit Hilfe von Delegates

[Beschreibung]

Beispiel

```
using System;
using System.Windows.Forms;
using System.Threading;

// Definition eines Delegat-Typs der Referenz auf Methode LongProcess() speichern kann
public delegate string LongProcessDelegate(string param);

public partial class Form1 : Form {
```

```
    LongProcessDelegate lpd; // Speichert Referenz auf Methode LongProcess()
    AsyncCallback acb; // Speichert Referenz auf Callback-Methode

    private void Form1_Load(object sender, EventArgs e) {
        // neue Instanz von LongProcessDelegate erzeugen, welche auf Methode LongProcess() zeigt.
        lpd = new LongProcessDelegate(LongProcess);
        // neue Instanz von AsyncCallback erzeugen, welche auf die Methode Callback() zeigt.
        acb = new AsyncCallback(Callback);
    }

    // Dies ist die Methode welche asynchron aufgerufen werden soll.
    public string LongProcess(string param) {
        Thread.Sleep(5000);
        return param.ToUpper();
    }

    // Callback-Methode (wird aufgerufen wenn LongProcess() beendet ist.
    // Die Callback-Methode muss ein Argument vom Typ IAsyncResult übernehmen.
    public void Callback(IAsyncResult ar) {
        // EndInvoke() von LongProcessDelegate aufrufen
        // dank des Parameters ar weiss lpd um welchen Methodenaufruf es sich handelt
        // es könnten nämlich mehrere pendent sein.
        string result = lpd.EndInvoke(ar);
        MessageBox.Show(result);
    }

    private void btnInvoke_Click(object sender, EventArgs e) {
        // Asynchroner Aufruf von LongProcess(), als zweites Argument wird ein
        // Delegat übergeben der auf die Callback-Methode zeigt.
        // BeginInvoke gibt ein IAsnycResult-Objekt zurück, mit dessen Eigenschaft
        // IsCompleted man abfragen könnte, ob LongProcess() bereits fertig ist.
        lpd.BeginInvoke(txtString.Text, acb, null);
    }

    public Form1() {
        InitializeComponent();
    }
}
```

15.2 Asynchroner Methodenaufruf mit BackgroundWorker

Unkomplizierter geht es mit der Komponente BackgroundWorker
Wichtige Member des BackgroundWorkers

Member	Beschreibung
`void` **`RunWorkerAsync`**`(object argument)`	Aufrufen um den asynchronen Vorgang zu starten. Alle benötigten Argumente im Parameter argument übergeben.
`event` **`DoWork`**`(object sender, DoWorkEventArgs e)`	Wird ausgelöst, wenn der BackgroundWorker den asynchronen Vorgang startet. Mit `e.Argument` kann man auf das in `RunWorkerAsync()` übergebene Objekt zugreifen. In diesem Event-Handler den Vorgang starten. Gibt der Vorgang einen Wert zurück, kann man diesen `e.Result` zuweisen.
`event` **`RunWorkerCompleted`**`(object sender, RunWorkerCompletedEventArgs e)`	Wird ausgelöst, wenn der BackgroundWorker den asynchronen Vorgang beendet hat. Auf den vom Vorgang zurückgegebenen Wert kann man mit `e.Result` zugreifen.

Beispiel:

```
private double LongRunningCalculation(double param) {
   // hier läuft eine Berechnung die lange dauert
}
private void btnStart_Click(object sender, EventArgs e) {
   backgroundWorker1.RunWorkerAsync(1000);
}
private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e) {
   e.Result =  LongRunningCalculation((double)e.Argument);
}
private void backgroundWorker1_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e) {
   MessageBox.Show("The Result is: " + e.Result.ToString());
}
```

15.3 Asynchrones Laden von Daten mit dem BackgroundWorker

Versuche, mit zwei Threads die Tabellen eines DataSets zu füllen sind bisher fehlgeschlagen. Die Daten wurden zwar geladen, doch die ComboBox an welche die mit dem BackgroundWorker geladenen Daten gebunden wurden zeigte keine Items an.
Definiert man jedoch ein zweites DataSet für die Daten welche im Hintergrund mit dem BackgroundWorker geladen werden wird es möglich. Der BackgroundWorker ruft eine Funktion auf, welche ein solches DataSet zurückgibt. Die Callback-Methode die im `BackgroundWorker_RunWorkerCompleted`-EventHandler aufgerufen wird übernimmt ein solches DataSet und bindet es an die ComboBox.

16 Assemblies und Verweise

Setzt man in einem Projekt A einen Verweis auf Assembly B, welche einen Verweis auf Assembly C hat, ist es nicht notwendig in Projekt A auch Assembly C zu referenzieren. Assembly C wird automatisch in das \bin-Verzeichnis von Projekt A kopiert.

17 Event-Handling

17.1 Delegates und Multicast-Delegates

Ein Delegate ist ein Stellvertreter für eine Methode. Ein Multicast-Delegate kann Stellvertreter für mehrere Methoden sein.
Definiert man ein Delegate mit Rückgabetyp `void`, so handelt es sich automatisch um ein Multicast-Delegate. Das heisst das Delegate kann Stellvertreter für mehrere Methoden sein.

Beispiel:

```
public delegate void TestDelegate(int number); // Deklaration eines Delegate-Typs

public partial class Form1 : Form {
        TestDelegate test1; // Deklaration eines Delegates vom Typ TestDelegate

        public Form1() {
                InitializeComponent();
                // test1 wird instanziiert und zeigt auf handler1
                test1 = new TestDelegate(handler1);

                // dem Delegate wird eine weitere Methode hinzugefügt
                test1 += handler2;
        }

        private void handler1(int num) {
                MessageBox.Show("handler1");
        }

        private void handler2(int num) {
                MessageBox.Show("handler2");
        }
        private void button1_Click(object sender, EventArgs e) {
                test1.Invoke(3); // dadurch werden die Methoden handler1 und handler2 aufgerufen
        }
}
```

18 Deployment

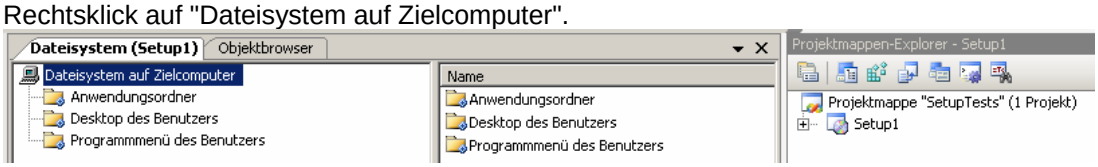
18.1 Installer Projekte

Windows Installer ist ein Service, welcher Installation, Updates, Reparaturen und Deinstallation von Applikationen managt. Windows Installer führt in einer Datenbank Buch über die Installation von Applikationen. Wird eine Anwendung deinstalliert, sorgt Windows Installer dafür, dass alle Registry-Einträge, Dateien auf der Harddisk, Icons im Startmenü und auf dem Desktop usw. wieder entfernt werden. Wird eine bestimmte Datei immer noch von einer anderen Applikation referenziert, wird sie vom Windows Installer nicht entfernt. Die Datenbank ermöglicht auch Reparaturen. Falls eine Registry-Einstellung oder eine DLL korrumpiert wird, kann man die Installation reparieren. Während der Reparatur liest der Installer die Datenbank und repliziert die Installation.

18.1.1 Assemblies im Global Assembly Cache installieren

Standardmässig werden die von der zu installierenden Applikation benötigten Assemblies ins Anwendungsverzeichnis kopiert. Um DLL's in den GAC zu installieren geht man folgendermassen vor:

1. Den Dateisystem-Editor auswählen.
2. Rechtsklick auf "Dateisystem auf Zielcomputer".



3. Den Kontextmenü-Eintrag "Cacheordner für globale Assembly" auswählen.
4. Danach erscheint im Dateisystem-Editor ein weiterer Ordner, welchem man mit Drag & Drop die im GAC zu installierenden Assemblies hinzufügen kann.

19 Pitfalls

19.1 Properties von Komponenten

Der Form-Designer von Visual Studio initialisiert alle Eigenschaften von Komponenten (d.h. Klassen welche mit Drag & Drop aus der Toolbox auf ein Formular gezogen werden) mit Standardwerten[2], auch wenn man im Eigenschaftenfenster keinen Wert zuweist. Dies auch wenn man das Attribut `[Browsable(false)]` auf die Eigenschaft anwendet. Siehe auch `DesignerSerializationVisibility` Attribut.
Daher ist bei der Programmierung des Set-Accessors darauf zu achten dass eine Initialisierung mit einem Standardwert wie 0 für int, `false` für bool, `null` für Referenztypen usw. dies nicht zu unerwünschtem Verhalten führt.

19.2 ValueMember

Vorsicht beim Setzen des ValueMembers von ComboBox (und ev. anderen Controls). Denn dabei wird bereits das `SelectedValueChanged` Ereignis ausgelöst. Will man im Event Handler einen Datenbankzugriff machen und setzt den ConnectionString erst in `DbInitialize()` kommt es natürlich zu einem Fehler.
Lösung:
Den Event-Handler erst am Schluss von `DbInitialize()` dynamisch hinzufügen.

19.3 OleDbDataReader

Allgemein bei DataReadern: diese nach Gebrauch immer schliessen, denn es wird jedes Mal eine Connection geöffnet. Ist die maximale Anzahl Connections überschritten, zeigt der Provider `Microsoft.Jet.OLEDB.3.51` die korrekte Fehlermeldung "Mehr Datenbanken können nicht geöffnet werden", während der neuere `Microsoft.Jet.OLEDB.4.0` die Meldung "Nicht erkennbares Datenbankformat" anzeigt.

20 Verschiedenes

20.1 Regular Expressions

Die Klassen für die Arbeit mit Regular Expressions befinden sich im Namespace `System.Text.RegularExpressions`.
Die wichtigste Klasse ist `Regex`. Sie übernimmt im Konstruktor einen Regular Expression als String.

20.1.1 Treffer durch anderen Text ersetzen

Die Methode `Replace()` übernimmt einen Input-String und einen String mit dem neuen Text (welcher die Treffer ersetzt). In diesem String kann man mit $& den Treffer referenzieren.
Beispiel:

```
Regex reg = new Regex("class");
MessageBox.Show(reg.Replace("public class Car", "<b>$&</b>"));
```

Der Aufruf von `reg.Replace()` gibt den folgenden String zurück:
`public <b>class</b> Car`.

20.2 E-Mail via Outlook versenden

```
System.Diagnostics.Process.Start("mailto:info@a1-it.ch?subject=Support&body=Ich habe ein Problem");
```

[2] Designergenerierter Code

Diese Codezeile startet Outlook, öffnet dort ein neues Mail und füllt die entsprechenden Werte für Empfänger, Betreff und Mail-Text ein.
Die Parameter `subject` und `body` können auch weggelassen werden.

20.3 Den Browser mit einer bestimmten URL aufrufen

```
System.Diagnostics.Process.Start("http://www.a1-it.ch");
```

Ruft den Standardbrowser des Systems mit der ensprechenden URL auf.

20.4 Standard-Drucker setzen

```
Imports System.Runtime.InteropServices

Public Class Form1
    <DllImport("winspool.drv")> _
    Private Shared Function SetDefaultPrinter(ByVal name As String) As Boolean
    End Function

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click
        SetDefaultPrinter("Automatisch HP LaserJet 4100 Series PCL auf PC-SUPPORT")
    End Sub
End Class
```

20.5 COM-Interop

20.5.1 Methoden mit optionalen Parametern aus C# aufrufen

C# kennt keine optionalen Parameter. VB.NET und COM hingegen schon. Im Namespace `System.Reflection` gibt es die Klasse `Missing`. Sie hat ein statisches Feld `Value`. Will man in C# einen optionalen Parameter nicht angeben, kann man `Missing.Value` benutzen. Falls die Methode den Parameter als Referenz will:

```
object oMissing = System.Reflection.Missing.Value;
comComponent.Methodenname(ref oMissing);
```

20.6 String in BitArray konvertieren

Diese Funktion konvertiert einen String in einen BitArray und verwendet dazu die im System eingestellte ANSI-Kodierung.

```
Imports System.Text
…
Public Function StringToBitArray(ByVal str As String) As BitArray
    Dim ansi As Encoding
    ansi = Encoding.Default
    Dim arrBytes() As Byte
    arrBytes = ansi.GetBytes(str)
    Dim arrBits As New BitArray(arrBytes)
    Return arrBits
End Function
```

20.7 Zeit und Datum formatieren

Mit der `ToString()` Methode der Struktur `DateTime` arbeiten, dieser kann man einen FormatString übergeben.
Beispiele:

FormatString	Ausgabe
`d.M.yy`	4.6.07
`d.M.yyyy`	4.6.2007
`dd.MM.yyyy`	04.06.2007
`dd.MMM.yyyy`	04.Jun.2007
`ddd, dd.MMM.yyyy`	Mo, 04.Jun.2007
`dddd, dd.MMMM.yyyy`	Montag, 04.Juni.2007
`h:mm`	6:21
`hh:mm`	06:21
`HH:mm`	18:21
`HH:mm:ss`	18:21:23

20.8 Präprozessordirektiven

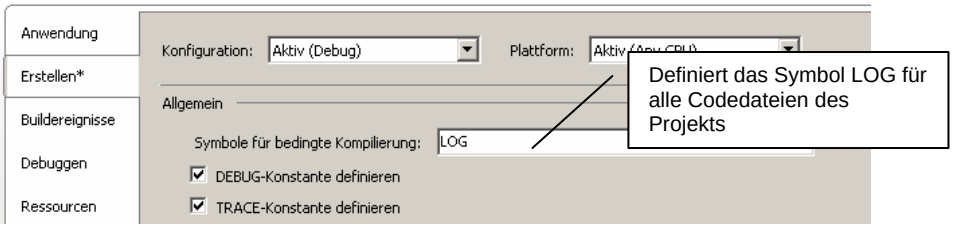
Im DEBUG-Modus sind automatisch die beiden Symbole DEBUG und TRACE definiert. Im RELEASE-Modus nur das Symbol TRACE.
Mit der Präprozessordirektive `#if <Symbol>` und `#endif` kann man einen Code-Block definieren, welcher nur kompiliert wird wenn ein bestimmtes Symbol gesetzt ist.

Beispiel:
```
#if DEBUG
    btnTest.Visible = true;
#endif
```

Neben den Symbolen DEBUG und TRACE kann man auch eigene Symbole definieren. Dazu benutzt man die Präprozessordirektive `#define`. `#define` muss jedoch immer an erster Stelle eines Codefiles stehen.

Beispiel:
```
#define LOG
```

In den Projekteinstellungen ist es möglich, Symbole für das gesamte Projekt zu setzen.



Projektweit gesetzte Symbole können für eine einzelne Code-Datei wieder ausgeschaltet werden. Dazu dient die Präprozessordirektive `#undef`. `#undef` muss ebenfalls in der Codedatei an erster Stelle stehen.

Beispiel:
```
#undef LOG
```

20.9 Tracing

Die Klasse `System.Diagnostics.Trace` verfügt über statische Member welche ermöglichen, Informationen zum Ablauf eines Programms in einen Textfile oder in das Windows Event Log zu schreiben.

Beispiel:
```
1 private void Form1_Load(object sender, EventArgs e) {
2       // der Listeners-Collection einen TextWriterTraceListener hinzufügen
3       // dieser schreibt in die Datei Tracing.txt
4       // ohne diese Zeile bewirkt Trace.WriteLine() keine Ausgabe
5       Trace.Listeners.Add(new TextWriterTraceListener("Tracing.txt", "A1-Textlistener"));
6       Trace.AutoFlush = true; // bewirkt, dass Trace.Flush() nicht aufgerufen werden muss
7       Trace.IndentSize = 2;
8       Trace.WriteLine("App gestartet");
9 }
```

Die TraceListeners können auch in der Konfigurationsdatei hinzugefügt werden. Dieser Eintrag im App.config bewirkt das gleiche wie die Zeilen 5 - 7 im Code

```
  <system.diagnostics>
    <trace autoflush="true" indentsize="2">
      <listeners>
        <add name="A1-Textlistener"
          type="System.Diagnostics.TextWriterTraceListener"
          initializeData="Tracing.txt" />
      </listeners>
    </trace>
  </system.diagnostics>
```

Für das Event Log muss man die Klasse `System.Diagnostics.EventLogTraceListener` verwenden.

20.9.1 Tracing mit Switches steuern

Durch das Hinzufügen eines Listeners in der Konfigurationsdatei können wir zwar das Tracing ein- und ausschalten. Wenn wir jedoch verschiedene Aspekte der Programmausführung verfolgen wollen (z.B. Performance, Datenbankzugriffe und User) wollen wir diese Aspekte unabhängig voneinander ein- und ausschalten können.
Die Methode `Trace.WriteLineIf(bool condition, string message)` ermöglicht, das Schreiben von Trace-Meldungen an eine Bedingung zu knüpfen. Was wir jetzt noch brauchen ist eine bool'sche Variable welche man über die App.config steuern kann. Diese Möglichkeit bietet die Klasse `System.Diagnostics.BooleanSwitch`.

```
<configuration>
  <system.diagnostics>
```

```
    <trace autoflush="true" indentsize="2">
      <listeners>
        <add name="myListener"
          type="System.Diagnostics.TextWriterTraceListener"
          initializeData="Trace.txt" />
      </listeners>
    </trace>
    <switches>
      <add name="myBooleanSwitch" value="1" /> <!-- 1 = true, 0 = false -->
    </switches>
  </system.diagnostics>
</configuration>
```

Mit dem Unterelement <add> des Elements <switches> können wir den Wert eines BooleanSwitch's steuern.
Im Code können wir nun einen damit verknüpften Switch instanziieren.

```
BooleanSwitch mySwitch = new BooleanSwitch("myBooleanSwitch", "Beschreibung");
```

Der erste Parameter des Konstruktors muss mit dem Attribut name des Elements <add> übereinstimmen.
Falls in der App.config kein entsprechender Switch definiert wird, erzeugt dies keinen Fehler. Der im Code instanziierte Switch ist ausgeschaltet.
Eine bedingte Trace Meldung lässt sich nun wie folgt schreiben:

```
Trace.WriteLineIf(mySwitch.Enabled, "Trace-Meldung");
```

20.10 Registry Zugriff

Die Klassen für den Zugriff auf die Windows-Registrierung befinden sich im Namespace `Microsoft.Win32`.

20.11 Pfade zu Systemordnern auslesen

Der Methode `Environment.GetFolderPath(Environment.SpecialFolder)` gibt den Pfad zu einem durch die Enumeration `Environment.SpecialFolder` spezifizierten Systemordner zurück.
`Environment.SpecialFolder` kann folgende Werte annehmen:

Membername	Beschreibung
ApplicationData	Ein Verzeichnis, das als allgemeines Repository für programmspezifische Daten des aktuellen Roamingbenutzers verwendet wird. Ein Roamingbenutzer arbeitet an mehreren Computern in einem Netzwerk. Das Profil eines Roamingbenutzers wird auf einem Server im Netzwerk gespeichert und in ein System geladen, wenn sich der Benutzer anmeldet.
CommonApplicationData	Das Verzeichnis, das als allgemeines Repository für programmspezifische Daten verwendet wird, die von allen Benutzern verwendet werden.
CommonProgramFiles	Das Verzeichnis für Komponenten, die von mehreren Anwendungen gemeinsam genutzt werden.
Cookies	Das Verzeichnis, das als allgemeines Repository für Internetcookies verwendet wird.
Desktop	Der logische Desktop und nicht der physikalische Speicherort im Dateisystem.
DesktopDirectory	Das Verzeichnis, das für das physikalische Speichern von Dateiobjekten auf dem Desktop verwendet wird. Dieses Verzeichnis darf nicht mit dem Desktopordner verwechselt werden, bei dem es sich um einen virtuellen Ordner handelt.
Favorites	Das Verzeichnis, das als allgemeines Repository für die Favoriten des Benutzers verwendet wird.
History	Das Verzeichnis, das als allgemeines Repository für die Internetverlaufselemente verwendet wird.

InternetCache	Das Verzeichnis, das als allgemeines Repository für temporäre Internetdateien verwendet wird.
LocalApplicationData	Das Verzeichnis, das als allgemeines Repository für programmspezifische Daten verwendet wird, die von einem aktuellen Benutzer verwendet werden, der kein Roamingbenutzer ist.
MyComputer	Der Ordner "Arbeitsplatz". **Hinweis** Die MyComputer-Konstante führt immer zu der leeren Zeichenfolge (""), da für den Ordner Arbeitsplatz kein Pfad definiert ist.
MyDocuments	Der Ordner "Eigene Dateien".
MyMusic	Der Ordner "Eigene Musik".
MyPictures	Der Ordner "Eigene Bilder".
Personal	Das Verzeichnis, das als allgemeines Repository für Dokumente verwendet wird.
ProgramFiles	Das Verzeichnis für Programmdateien.
Programs	Das Verzeichnis, das die Programmgruppen des Benutzers enthält.
Recent	Das Verzeichnis, das die vom Benutzer zuletzt verwendeten Dokumente enthält.
SendTo	Das Verzeichnis, das die Elemente für das Menü Senden an enthält.
StartMenu	Das Verzeichnis, das die Elemente für das Menü Start enthält.
Startup	Das Verzeichnis, das der Programmgruppe Autostart des Benutzers entspricht. Diese Programme werden vom System gestartet, sobald ein Benutzer Windows NT oder eine neuere Version startet oder sich dort anmeldet oder wenn er Windows 98 startet.
System	Das Verzeichnis System.
Templates	Das Verzeichnis, das als allgemeines Repository für Dokumentvorlagen verwendet wird.

21 Grafik

21.1 Bilder in MySql speichern

Beispiel:

```
private void button1_Click(object sender, EventArgs e) {
    Image image = Image.FromFile(@"..\..\..\DominikHug.jpg");
    MemoryStream mstream = new MemoryStream();
    image.Save(mstream, ImageFormat.Jpeg);

    taUsers.FillByID(dsPicture.tblusers, 51);
    dsPicture.tblusers[0].Bild = mstream.ToArray();
    taUsers.Update(dsPicture.tblusers);
}

private void button2_Click(object sender, EventArgs e) {
    taUsers.FillByID(dsPicture.tblusers, 51);
    MemoryStream mstream = new MemoryStream();
    mstream.Write(dsPicture.tblusers[0].Bild, 0, dsPicture.tblusers[0].Bild.Length);
```

```
    Image image = Image.FromStream(mstream);
    pictureBox1.Image = image;
}
```

22 Steuerung von Office

22.1 Word

Für den Zugriff auf Word 2003 einen Verweis auf `Microsoft Word 11.0 Object Library` setzen (COM).
Ein Objekt des Typs `Microsoft.Office.Interop.Word.Application` erzeugen.
Setzt man deren `Visible` Eigenschaft auf `true`, wird Word sichtbar.
Mit `Documents.Open()` kann man ein Dokument öffnen. Diese Methode verlangt 16 ref Parameter vom Typ Object. Daher muss man die Parameter zuerst als Object-Variablen erzeugen und mit dem Schlüsselwort `ref` übergeben.

Beispiel:

```
Microsoft.Office.Interop.Word wordApp = new Microsoft.Office.Interop.Word.Application();
wordApp.Visible = true;
// diese beiden Variablen müssen vom Typ object sein
// und im Voraus deklariert werden, damit sie mit ref übergeben werden können
object oMissing = System.Reflection.Missing.Value;
object docPath = @"Baustellenabnahmeprotokoll.dot";
// ein Dokument öffnen
wordApp.Documents.Open(ref docPath, ref oMissing, ref oMissing, ref oMissing, ref oMissing, ref
oMissing, ref oMissing, ref oMissing, ref oMissing, ref oMissing, ref oMissing, ref oMissing, ref
oMissing, ref oMissing, ref oMissing, ref oMissing);
```

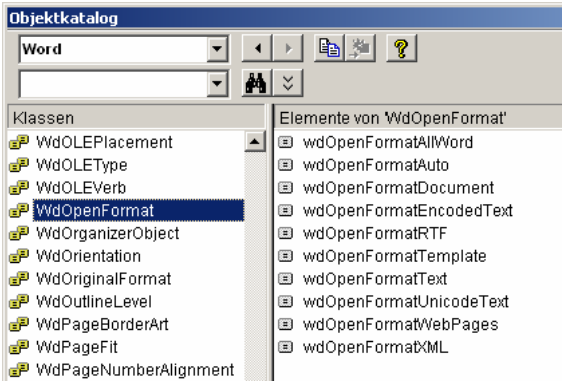
22.1.1 Arbeiten mit Textmarken

Beispiel: Prüfen ob eine Textmarke existiert, fall ja Text einfügen

```
object bmDatum = "Datum"; // Datum ist der Name der Textmarke
// die Methode Exists() verlangt einen Parameter vom Typ string, deshalb bmDatum.ToString()
if (wordApp.ActiveDocument.Bookmarks.Exists(bmDatum.ToString())) {
    wordApp.ActiveDocument.Bookmarks.get_Item(ref bmDatum).Select();
    wordApp.Selection.TypeText("plapperlapapp");
}
```

22.1.2 Arbeiten mit den Enumerationen von Office

VBA definiert eine Vielzahl von Enumerationen. Meist benötigt man diese als Parameter von Methoden des Office-Objektmodells. Aus .NET Applikationen müssen sie als `ref` Parameter vom Typ `object` übergeben werden.



Deswegen ist es notwendig, zuerst eine Variable des Typs `object` zu deklarieren und ihr ein Enumerationswert zuzuweisen. Nur so ist es möglich, den Parameter als `object` mit `ref` zu übergeben.

Beispiel:

```
object filePath = @"C:\Dokumente\Testdokument.doc";
object format = Word.WdOpenFormat.wdOpenFormatDocument;
object oMissing = System.Reflection.Missing.Value;
wordApp.Documents.Open(ref filePath, ref oMissing, ref oMissing, ref oMissing, ref oMissing,
ref oMissing, ref oMissing, ref oMissing, ref oMissing, ref format, ref oMissing, ref oMissing,
ref oMissing, ref oMissing, ref oMissing, ref oMissing);
```

22.2 Outlook

Als erstes einen Verweis auf `Microsoft Outlook 11 Object Library setzen`.

22.2.1 Zugriff auf Kontake

Kontakt im Standard-Kontakt Ordner erzeugen:

```
Microsoft.Office.Interop.Outlook.Application app;
Microsoft.Office.Interop.Outlook.NameSpace ns;
app = new Microsoft.Office.Interop.Outlook.Application();
ns = app.GetNamespace("MAPI");
ns.Logon(null, null, false, false);

ContactItem contact = (ContactItem)app.CreateItem(OlItemType.olContactItem);
contact.LastName = "Müller";
contact.FirstName = "Peter";
contact.Email1Address = "peter.mueller@hotmail.com";
contact.Save();
```

Einige weitere Properties von ContactItem

- BusinessAddressStreet
- BusinessAddressCity
- BusinessAddressPostalCode
- PrimaryTelephoneNumber
- MobileTelphoneNumber

Kontakte in einem DataGridView auflisten

```
Microsoft.Office.Interop.Outlook.Application app;
Microsoft.Office.Interop.Outlook.NameSpace ns;
Microsoft.Office.Interop.Outlook.MAPIFolder contactsFolder;
app = new Microsoft.Office.Interop.Outlook.Application();
ns = app.GetNamespace("MAPI");
ns.Logon(null, null, false, false);
contactsFolder = ns.GetDefaultFolder(OlDefaultFolders.olFolderContacts);
List<Contact> lstContact = new List<Contact>();
for (int i = 1; i <= contactsFolder.Items.Count; i++) {
    Microsoft.Office.Interop.Outlook.ContactItem item;
        item = (Microsoft.Office.Interop.Outlook.ContactItem)contactsFolder.Items[i];
    Contact contact = new Contact();
    contact.FamilyName = item.LastName;
    contact.ForeName = item.FirstName;
    contact.Email1 = item.Email1Address;
    lstContact.Add(contact);
}
dgvContacts.DataSource = lstContact;
...
public class Contact {
    string _foreName;
    public string ForeName {
        get { return _foreName; }
        set { _foreName = value; }
    }

    string _familyName;
    public string FamilyName {
        get { return _familyName; }
        set { _familyName = value; }
    }

    string _email1;
    public string Email1 {
        get { return _email1; }
        set { _email1 = value; }
    }
}
```

Unterordner vom Kontaktordner auflisten:

```
...
contactsFolder = ns.GetDefaultFolder(OlDefaultFolders.olFolderContacts);
for (int i = 1; i <= contactsFolder.Folders.Count; i++ ) {
    subFolder = contactsFolder.Folders[i]; // Indizes beginnen mit 1
    MessageBox.Show(subFolder.Name);
}
```

22.2.2 Zugriff auf EMails

Alle Emails eines Absenders in einem DataGridView auflisten

```
app = new Microsoft.Office.Interop.Outlook.Application();
ns = app.GetNamespace("MAPI");
ns.Logon(null, null, false, false);
```

```
inboxFolder = ns.GetDefaultFolder(OlDefaultFolders.olFolderInbox);
List<Mail> lstMails = new List<Mail>();
for (int i = 1; i <= inboxFolder.Items.Count; i++) {
    Microsoft.Office.Interop.Outlook.MailItem item =
(Microsoft.Office.Interop.Outlook.MailItem)inboxFolder.Items[i];
    if (item.SenderEmailAddress == "albert.balogh@a1-it.ch") {
        Mail mail = new Mail();
        mail.SenderEmail = item.SenderEmailAddress;
        mail.Subject = item.Subject;
        mail.Timestamp = item.ReceivedTime;
        lstMails.Add(mail);
    }
}
dgvMails.DataSource = lstMails;
...
public class Mail {
    string _subject;
    public string Subject {
        get { return _subject; }
        set { _subject = value; }
    }

    string _senderEmail;
    public string SenderEmail {
        get { return _senderEmail; }
        set { _senderEmail = value; }
    }

    DateTime _timestamp;
    public DateTime Timestamp {
        get { return _timestamp; }
        set { _timestamp = value; }
    }
}
```

23 Internet Zugriff

23.1 HTTP

23.1.1 HTML einer Webseite herunterladen

Beispiel:

```
using System.Net;
using System.IO;
…
WebClient webClient = new WebClient();
Stream stream = webClient.OpenRead("http://www.agri-shop.ch/login.php");
StreamReader sr = new StreamReader(stream, Encoding.UTF8);
txtHtml.Text = sr.ReadToEnd(); // HTML in TextBox anzeigen
stream.Close();
```

23.2 FTP

23.2.1 File Upload

Beispiel:

```
using System.Net;
using System.IO;
…
FtpWebRequest request =
(FtpWebRequest)WebRequest.Create("ftp://www.balogh.ch/FTP/HandguideMySQL.doc");
request.Method = WebRequestMethods.Ftp.UploadFile;

// hochzuladende Datei in Byte-Array verpacken
FileStream sourceStream = new FileStream("C:\\HandguideMySQL.doc", FileMode.Open);
byte[] fileContents = new byte[sourceStream.Length];
sourceStream.Read(fileContents, 0, Convert.ToInt32(sourceStream.Length));
sourceStream.Close();

request.ContentLength = fileContents.Length;
request.Credentials = new NetworkCredential("user1", "Password1");
request.KeepAlive = false;
Stream requestStream = request.GetRequestStream();
requestStream.Write(fileContents, 0, fileContents.Length);
```

```
requestStream.Close();
FtpWebResponse response = (FtpWebResponse)request.GetResponse();
MessageBox.Show(response.StatusDescription);
response.Close();
```

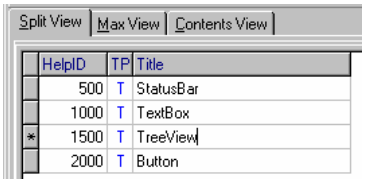
24 DotCHM

24.1 Hilfe-Projekte

DotCHM verwaltet die Projekte in einer eigenen Datenbank. Die bisher einzig bekannte Methode, ein Projekt auf einen anderen Computer zu verschieben / kopieren besteht darin, das Installationsverzeichnis (C:\Programme\DotCHM) zu verschieben.

24.2 Neues Help Topic hinzufügen

Menü Insert | Topic. In der Split View den Titel einfügen.



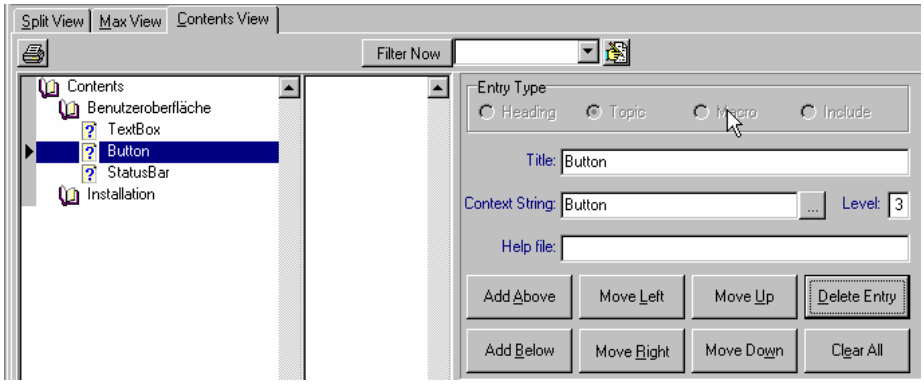
24.2.1 Weitere Keys für das Help Topic

Der Titel des Help Topics ist automatisch ein Key dafür und erscheint im .CHM File unter Index. Um weitere Keys hinzuzufügen gibt man in der Split View oder der Max View im Feld Keys weitere Indexwörter ein und trennt sie durch Semikola.

24.3 Contents organisieren (Inhaltsverzeichnis)

24.3.1 Ein Topic ins Inhaltsverzeichnis aufnehmen

Zur Contents View schalten.

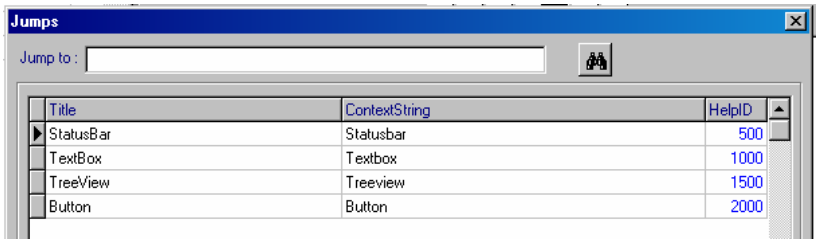


Falls bereits Einträge vorhanden, einen auswählen und den Button "Add Above" oder "Add Below" betätigen. Es erscheint ein Dialog aus welchem man durch Doppelklick ein Topic auswählen kann.

24.4 Hyperlinks

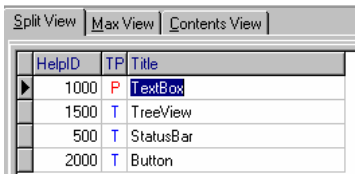
24.4.1 Topic Jumps

Ein Topic Jump ist ein Hyperlink auf ein anderes Topic im HelpFile. Zuerst wählt man das entsprechende Wort im Text und klickt auf das Icon Topic Jump (weisse Hand). Im erscheinenden Dialog kann man das Ziel-Topic auswählen.



24.4.2 Popup Jumps

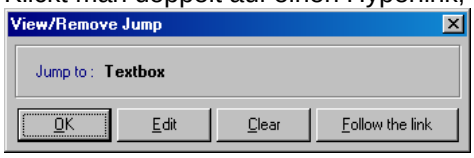
Ein Popup Jump ist ein Hyperlink bei welchem auf Klick ein kleinen Popup-Fenster erscheint. Um einen Popup Jump zu erzeugen benötigt man zuerst ein Topic vom Typ Popup. Diese kann man wie normale Topics erstellen, allerdings muss man in der Split View den Typ von T auf P ändern.



Das Topic TextBox ist jetzt vom Typ Popup. Man kann dieses Popup nun in einen anderen Text einfügen indem man einen Bereich markiert und auf das Symbol Popup Jump klickt. Es erscheint der gleiche Dialog wie beim Topic Jump, es werden aber nur die Topic vom Typ Popup angezeigt.

24.4.3 Hyperlinks löschen

Klickt man doppelt auf einen Hyperlink, erscheint dieser Dialog.



24.5 Integration von .CHM-Dateien in eine .NET Applikation

- Eine HelpProvider Komponente auf das Formular oder das Custom Control ziehen.
- HelpProvider.HelpNamespace auf die .CHM-Datei setzen.
- Das Formular und alle seine Controls erhalten nun drei neue Eigenschaften:
  - HelpKeyword auf HelpProvider
  - HelpNavigator auf HelpProvider
  - HelpString auf HelpNavigator
- Für jedes Control, das ein eigenes Topic hat, "HelpNavigator auf HelpProvider" auf TopicId setzen.
- HelpKeyword auf HelpProvider auf die TopicId setzen.

Drückt der User nun F1, erscheint das HelpTopic für das Control welches momentan den Fokus hat.

Mann kann die .CHM-Datei auch Code-gesteuert mit einer bestimmten TopicId aufrufen.

Beispiel

```
Help.ShowHelp(Me, "Help.chm", HelpNavigator.TopicId, "1200")
' Wichtig: TopicId als String übergeben
```

Siehe auch: HelpButton Eigenschaft

25 Visual CHM

Eine weitere Software um Hilfedateien im CHM-Format zu erstellen ist Visual CHM. Es ist Shareware und kann über www.vchm.com bezogen werden. Bei der Trial Version kann man nur 15 HTML-Seiten in eine CHM-Datei kompilieren. Es zu lizenzieren kostet USD 19.95.
Im Gegensatz zu DotCHM verwaltet Visual CHM die Projekte in Dateien. Es ist also kein Problem ein Hilfeprojekt von einem Computer auf einen anderen zu kopieren oder ein Backup zu erstellen (was mit DotCHM leider nicht so einfach möglich ist).

25.1 Integration in .NET Applikationen

Im Gegensatz zu DotCHM gibt es bei Visual CHM keine TopicId's.
Vorgehen:

1. HelpProvider auf das Formular oder das UserControl ziehen

2. `HelpNamespace` Eigenschaft des HelpProviders auf den Pfad zur CHM Datei setzen (absoluter oder relativer Pfad)
3. Bei den Controls zu welchen Hilfe verfügbar ist:
4. Eigenschaft `HelpNavigator auf helpProvider` auf "Topic" setzen
5. Eigenschaft `HelpKeyword auf helpProvider` auf den Namen der betreffenden HTML-Datei setzen (es ist auch möglich mit <Dateiname>#<Anker> zu einer bestimmten Stelle im HTML Dokument zu springen).

26 Ideen

26.1 Typisierte DataSets erweitern

Die in typisierten DataSets generierten Klassen die von DataTable und von DataRow abgeleitet sind, sind im generierten Code als partial deklariert. Sie können also um weitere Member ergänzt werden. So könnte man z.B. einer tbl_KundeDataRow einige weitere Member geben und hätten somit direkt eine wunderbar zu bindende Business Class.

27 Index