

Kochen mit Patrick

Achtung, diesen Monat ist der Restaurantkritiker unterwegs. Nichts bleibt an seinem Platz, alles wird argwöhnisch beäugt und hinterfragt. Das geschieht nicht ohne Grund, denn es lässt sich vieles verbessern und optimieren – auch in Ihrer Küche!



Patrick A. Lorenz ist Geschäftsführer der PGK GmbH, einem auf .NET spezialisierten Technologie-dienstleister. Daneben ist er als Autor tätig. Sein neuestes Buch „ASP.NET 3.5 mit AJAX“ beschreibt die Neuerungen in .NET 3.5 für Webentwickler. In seiner Freizeit ist Patrick Hobbykoch. Sie erreichen ihn unter www.pgk.de und lorenz@pgk.de.

Bereits in [1] waren die Neuerungen in C# 3.0 Thema dieser Kolumne. Inzwischen ist die neue Version zusammen mit Visual Studio 2008 ganz offiziell veröffentlicht worden. Zu den Neuerungen gehören unter anderem Erweiterungsmethoden, die es erlauben, bestehende Klassen virtuell um zusätzliche Methoden zu ergänzen. Sie können zum Beispiel für Datumsangaben eine Methode *ToConsole* definieren:

```
public static class DateTimeExtensions {
    public static void ToConsole(this DateTime dt)
    {
        Console.WriteLine(dt.ToString());
    }
}
```

Durch die Markierung des ersten Parameters mit dem Schlüsselwort *this* wird die Methode

vom Compiler und der Entwicklungsumgebung als Erweiterung des angegebenen Datentyps *DateTime* angesehen und kann daher virtuell auf diesen angewandt werden, sobald der Namensraum mit der Erweiterungsmethode referenziert wird:

```
var today = DateTime.Now;
today.ToConsole();
```

Tatsächlich ist der Aufruf nur eine bequeme Abkürzung für folgende Zeile:

```
DateTimeExtensions.ToConsole(today);
```

Der Vorteil der Erweiterungsmethoden besteht primär in der guten Lesbarkeit und der kontextuellen Bereitstellung der Methoden zum Beispiel mittels IntelliSense. In der Praxis können Sie mit Erweiterungsmethoden gerade bei immer wiederkehrenden Hilfskonstrukten und -abfragen jede Menge Zeit und Tipparbeit sparen und Ihren Quelltext darüber hinaus deutlich übersichtlicher gestalten.

Ein typisches Beispiel ohne Extension Methods

Wenn Sie hin und wieder mit ASP.NET Webanwendungen entwickeln, dann wird Ihnen der Ansatz des folgenden Beispiels bestimmt bekannt vorkommen. Die Seite enthält ein *Repeater*-Control mit dessen Hilfe eine Liste von Personen ausgegeben wird, wie sie in Abbildung 1 zu sehen ist:

```
<asp:Repeater ID="RP" runat="server">
  <HeaderTemplate><ul></HeaderTemplate>
  <ItemTemplate>
  <li>
    <asp:HyperLink id="HL" runat="server" />
    <asp:Literal ID="LT_BirthDate"
      runat="server" />
  </li>
  </ItemTemplate>
  <FooterTemplate></ul></FooterTemplate>
</asp:Repeater>
```

Zur Ausgabe der heterogenen Daten reicht eine gewöhnliche deklarative Bindung nicht, sondern Sie benötigen zusätzliche Funktionalität, welche die Daten im *ItemDataBound*-Ereignis des *Repeater*-Controls aufbereitet. Listing 1 zeigt, wie das aussehen könnte. Ganz schön viel Code für eine derart simple Darstellung. Es geht aber

auch anders! Im Folgenden zeige ich Ihnen, wie Sie Ihren Aufwand halbieren können.

Codeoptimierung mit Extension Methods

Mit Erweiterungsmethoden können Sie den Tippaufwand drastisch reduzieren – fast jede Zeile bietet dazu Potenzial, so auch gleich die erste:

```
if((e.Item.ItemType == ListItemType.Item) ||
    (e.Item.ItemType ==
        ListItemType.AlternatingItem)) {
```

Ein Klassiker, der in manchen Webanwendungen sicherlich hunderte Male auftaucht. Mit einer einfachen Erweiterung können Sie die Zeile wie folgt reduzieren:

```
if(e.Item.IsDataItem()) {
```

Die dazu notwendige Erweiterungsmethode sieht so aus:

```
public static bool IsDataItem(this
    RepeaterItem item) {
    return ((item.ItemType ==
        ListItemType.Item) ||
        (item.ItemType ==
            ListItemType.AlternatingItem));
}
```

Es folgt die Konvertierung des in den Ereignisargumenten übergebenen Datenelements:

```
var person = (e.Item.DataItem as Person);
```

Auch dies lässt sich mittels einer generischen Erweiterungsmethode optimieren:

```
var person = e.Item.Get<Person>();
```

```
public static T Get<T>(this
    IDataItemContainer item)
    where T : class {
    return (item.DataItem as T);
}
```

Listing 1

Code zur Ausgabe der heterogenen Daten.

```
protected void RP_ItemDataBound(object sender, RepeaterItemEventArgs e) {
    if((e.Item.ItemType == ListItemType.Item) ||
        (e.Item.ItemType == ListItemType.AlternatingItem)) {

        var person = (e.Item.DataItem as Person);
        var name = new StringBuilder();
        name.Append(person.LastName);
        if(person.Title != null) name.AppendFormat(", {0}", person.Title);
        if(person.FirstName != null) name.AppendFormat(", {0}", person.FirstName);

        var hl = (e.Item.FindControl("HL") as HyperLink);
        if(hl != null) {
            hl.Text = name.ToString();
            hl.NavigateUrl = string.Format("/person.aspx?id={0}", person.ID);
        }
        var lt = (e.Item.FindControl("LT_BirthDate") as Literal);
        if(lt != null) {
            if((person.BirthDate != DateTime.MinValue) &&
                (person.BirthDate != DateTime.MaxValue)) {
                lt.Text = string.Format("geboren am {0:d}", person.BirthDate);
            }
            else
            {
                lt.Text = "(Geburstag unbekannt)";
            }
        }
    }
}
```

Richtig viel Tipparbeit lässt sich bei der im Beispiel gezeigten kommaseparierten Verknüpfung von mehreren Zeichenketten einsparen.

In Abbildung 1 ist zu erkennen, dass die Zeichenketten teilweise optional sind. So wurde für eine Person ein akademischer Titel hinterlegt und für die andere nicht. Statt der vier Zeilen im Original lässt sich die Verknüpfung in optimierter Form auch in einer einzigen unterbringen:

```
var name = person.LastName.Append(", ",
    person.Title, person.FirstName);
```

Die dazu benötigte Erweiterungsmethode nimmt per *params*-Parameter eine beliebige Anzahl Strings entgegen und trennt diese mit dem angegebenen Separator.

```
public static string Append(this
    string input, string separator,
    params string[] values) {

    var sb = new StringBuilder();
    sb.Append(input);
    foreach(var value in values) {
        if(!string.IsNullOrEmpty(value)) {
            sb.Append(separator);
            sb.Append(value);
        }
    }
    return sb.ToString();
}
```

Ein absoluter Klassiker im Code Behind einer ASP.NET-Seite ist die Verwendung von *FindControl* zur Ermittlung einer benannten Control-Instanz:

```
var hl = (e.Item.FindControl("HL") as
    HyperLink); if(hl != null) {
```

Das Ergebnis wird als Control zurückgeliefert, muss daher konvertiert und am besten noch auf Gültigkeit geprüft werden. Mit einer generischen Erweiterungsmethode,



[Abb. 1] Der zur Ausgabe benötigte Quelltext lässt sich radikal reduzieren.

einem Predicate [2] und einem Lambda-Ausdruck lassen sich typisierte Abfragen mitsamt bedingter Ausführung sehr kompakt formulieren:

```
e.Item.FindControlExecute<HyperLink>("HL",
hl => {
hl.Text = name;
hl.NavigateUrl =
string.Format("/person.aspx?id={0}",
person.ID);
});
```

Die Logik findet sich in einer Erweiterungsmethode für den Datentyp *Control* wieder. Als Parameter für den Lambda-Ausdruck ist das generische Delegate *Action<T>* angegeben, das eine Methode mit einem Parameter und ohne Rückgabewert definiert.

```
public static bool
FindControlExecute<T>(this
Control control, string id,
Action<T> action) where T : Control {
var ctl = (control.FindControl(id) as T);
if(ctl != null) {
action((T) ctl);
return true;
}
return false;
}
```

Damit bleiben zur Optimierung nur noch die beiden Formatierungsanweisungen für die URL sowie die bedingte Ausgabe des Geburtstags. Es bietet sich an, eine möglichst allgemeingültige *Format*-Methode für alle Arten von Wertetypen zu implementieren, also eine Abkürzung für *string.Format*. Deren Anwendung könnte so aussehen – beachten Sie, dass die For-

matierung direkt der *Format*-Methode des Datentyps *Guid* übergeben wird:

```
hl.NavigateUrl =
person.ID.Format("/person.aspx?id={0}");
```

Während sich hier wenig Quelltext einsparen ließ, sieht es bei der bedingten Zuweisung des Datums schon ganz anders aus. Hier kommt eine neue Methode namens *FormatIf* zum Einsatz, der sowohl die Formatierungsanweisung als auch ein alternativer Text übergeben werden kann:

```
lt.Text = person.BirthDate.FormatIf(
"geboren am {0:d}", "(Geburtstag
unbekannt)");
```

Die verwendete Erweiterungsmethode *FormatIf* ist mehrfach überladen und steht auf allen Wertetypen zur Verfügung.

```
public static string
FormatIf<T>(this T value,
Func<T, bool> condition, string format,
string defaultText) where T : struct {
if(condition == null)
condition = v => v.IsNotNull();
return (condition(value) ?
string.Format(format, value) :
defaultText);
}
```

Die Methode prüft den übergebenen Wert mithilfe einer ebenfalls übergebenen Bedingung. Diese ist in Form eines Delegates vom Typ *Func<T, bool>* definiert, also einer Methode mit einem Parameter vom Typ *T* (dem zu prüfenden Wert) und einem booleschen Rückgabewert. Trifft die Bedingung zu, wird die Formatierung ausgeführt

und ansonsten der Alternativtext zurückgeliefert. Sofern keine Bedingung übergeben wurde, wird *IsEmpty* verwendet. Dabei handelt es sich ebenfalls um eine generische Erweiterungsmethode für alle Wertetypen, die zusammen mit *IsEmpty* prüft, ob der Wertetyp einen Wert enthält:

```
public static bool IsEmpty<T>(this
T value) where T : struct {
return value.Equals(default(T));
}
public static bool IsNotEmpty<T>(this
T value) where T : struct {
return (value.IsEmpty() == false);
}
```

Das große Extension Method Makeover

Das praxisnahe Originalbeispiel aus dem Listing 1 hat satte 16 Zeilen unmittelbaren Quelltext benötigt. Mithilfe von Erweiterungsmethoden kann der Code nicht nur deutlich übersichtlicher gestaltet werden, sondern auch wesentlich kompakter. Listing 2 zeigt das komplette, überarbeitete Beispiel mit jetzt nur noch acht Zeilen effektivem Code, also genau der Hälfte.

Noch mehr Helfer

In den Quelltexten zu diesem Artikel finden Sie noch einige weitere Helferlein für den täglichen Bedarf. Sehr hilfreich sind vereinfachte Zeichenkettenoperationen wie die folgenden Methoden. *string.HasText* prüft zum Beispiel, ob eine Zeichenkette nicht *null* ist und nach dem Abschneiden von Leerzeichen (*Trim*) noch Text beinhaltet.

Komplexe Operationen lassen sich mit Hilfsmethoden für reguläre Ausdrücke realisieren. So kann eine Zeichenkette auf eine E-Mail-Adresse hin geprüft werden:

```
var email = "lorenz@pgk.de";
var isEmail =
email.IsMatch(@"^[w\.\-]+
@([w\-\+\.]*[w\-\-]{2,63}\
[a-zA-Z]{2,4}$");
```

Um aus einer Zeichenkette Werte zu extrahieren und gleich zu konvertieren, verwenden Sie die *Extract*-Methode samt Lambda-Ausdruck:

```
var numbers = "123, 456, 789";
foreach(int number in
numbers.Extract(@"\d+", n =>
int.Parse(n))) {
Console.WriteLine(number);
}
```

Ähnlich komfortabel können Sie auch komplexe Ersetzungen durchführen, etwa

Listing 2

Der optimierte Code.

```
protected void RP_ItemDataBound(object sender, RepeaterItemEventArgs e) {
if(e.Item.IsDataItem()) {
var person = e.Item.Get<Person>();
var name = person.LastName.Append(", ", person.Title, person.FirstName);
e.Item.FindControlExecute<HyperLink>("HL", hl => {
hl.Text = name;
hl.NavigateUrl = person.ID.Format("/person.aspx?id={0}");
});
e.Item.FindControlExecute<Literal>("LT_BirthDate", lt =>
lt.Text = person.BirthDate.FormatIf("geboren am {0:d}",
"(Geburtstag unbekannt)");
});
}
}
```

um HTML-Auszeichnungen (und nur diese) in Kleinbuchstaben zu konvertieren:

```
var html = "<STRONG>Text</STRONG>";
var htmlLower = html.Replace("<.+?>",
    m => m.Value.ToLower());
```

Auf die Eigenschaften beliebiger Objekte greifen Sie mithilfe der beiden Methoden `GetProperty` und `SetProperty` zu:

```
var person = new Person { FirstName =
    "Max", LastName = "Müller" };
var lastName =
    person.GetProperty<string>(
    "LastName");
Console.WriteLine(lastName);
```

Die Moral von der Geschicht'

Insgesamt wurde freilich deutlich mehr Code neu geschrieben als eingespart. Die neuen Erweiterungsmethoden sind aller-

dings so generisch, dass sie sich vom Fleck weg projektübergreifend einsetzen lassen. Wie im Beispiel lassen sich Optimierungen am besten im Rahmen eines Code-Reviews ermitteln. Gehen Sie einfach typische Beispiele Ihrer eigenen Quelltexte durch und überlegen Sie, wo Sie mit wiederkehrenden Konstrukten letztlich viel Tipparbeit investieren müssen.

Zur Wiederverwendung von Erweiterungsmethoden bietet sich die Aufwertung eines bereits bestehenden Utility-Projekts an oder die Anlage eines neuen Projektes speziell für die Extensions. Verwenden Sie dabei einen möglichst hohen Namensraum (zum Beispiel *PGK*), stehen die Methoden in anderen Projekten (zum Beispiel *PGK.Kunde.Projekt*) automatisch zur Verfügung, wenn Sie die Assembly referenzieren.

Übrigens: Anders als einige andere Spracherweiterungen von C# 3.0 benöti-

gen die Erweiterungsmethoden tatsächlich das .NET Framework 3.5. Dies hat mit der internen Abbildung zu tun. Der Compiler setzt diese als ganz „normale“ statische Methoden um, kennzeichnet diese jedoch mithilfe des speziellen Attributs

```
System.Runtime.CompilerServices.
    ExtensionAttribute
```

Auf diese Weise können Compiler und Entwicklungsumgebung die Methoden erkennen und deren Verwendung zum Beispiel mittels IntelliSense anbieten. **[bl]**

[1] Patrick A. Lorenz, Kochen mit Patrick, dotnetpro 2/2008, S. 115ff.

[2] Patrick A. Lorenz, Kochen mit Patrick, Predicates benutzen, dotnetpro 3/2008, S. 110f.

Fisch statt Fleisch

Die Rezepte der vorangegangenen Ausgaben waren ausgesprochen fleischlastig. Diesmal gibt's deshalb, wie angekündigt, ein Gericht ganz ohne Fleisch.



Lachs mit Kräuterkruste

Der Lachs ist quasi ein Standardfisch. Er gehört aber auch zu den ganz wenigen Arten, die von Umweltorganisationen nicht als bedroht ausgewiesen werden und die man daher noch mit gutem Gewissen essen kann. Dies gilt allerdings ausschließlich für Alaska-Wildlachs – der ebenfalls empfohlene Seelachs ist eigentlich ein Dorsch. Lachs lässt sich in unzähligen Variationen zubereiten, eine abwechslungsreiche und leckere davon kommt mit Kräuterkruste aus dem Ofen.

Für die Kruste kaufen Sie am besten auf dem Wochenmarkt ein paar Bund ganz frische gemischte Kräuter, zum Beispiel Rosmarin und Thymian, aber auch Petersilie und Koriander. Je mehr Kräuter, desto besser. Diese schneiden Sie fein (nicht hacken) und geben Sie in eine Schüssel. Dazu kommen drei, vier ebenfalls fein geschnittene Frühlingzwiebeln sowie 100 g Semmelbrösel und 50 g zerlassene Butter. Nachdem Sie eine Prise Salz und frisch gemahlene Pfeffer dazu gegeben haben, mischen Sie die Zutaten gut durch.

Verteilen Sie drei bis vier Lachsfilets von je etwa 150 g auf einem Backblech und schneiden Sie diese jeweils dreimal schräg ein. Reiben Sie den Fisch anschließend mit der Kräuterkruste ein. Sie darf ruhig dick sein und sollte fest angedrückt werden. Anschließend geht's bei 180 Grad für rund 20 Minuten in den Ofen. Guten Appetit!