

17

Kulturabhängiges Formatieren von Zahlen- und Datumswerten

497	Allgemeines über Format Provider in .NET
498	Kulturabhängige Formatierungen mit CultureInfo
501	Formatierung durch Formatzeichenfolgen
511	Gezielte Formatierungen mit Format Providern
513	Kombinierte Formatierungen
517	So helfen Ihnen benutzerdefinierte Format Provider, Ihre Programme zu internationalisieren

Beim Durcharbeiten der vergangenen Kapitel sind Sie bereits auf die eine oder andere Funktionalität von .NET gestoßen, Zahlen oder Zeitwerte aufzubereiten, um sie in bestimmten Darstellungsformaten auszugeben. Auch die Vorgehensweise zur Vorgabe von Zeichenmustern für das Parsen von Zeichenketten, um sie in entsprechende Datentypen umzuwandeln, konnten Sie schon in einigen Beispielen kennen lernen. Doch glauben Sie mir: Sie haben bislang nur die Spitze des Eisberges gesehen.

Die folgenden Abschnitte beschäftigen sich intensiver mit der textlichen Aufbereitung von Daten, und insbesondere der Umgang mit den verfügbaren Format Providern bis hin zur Realisierung eigener Format Provider wird für Sie sicherlich von großen Interesse sein.

Allgemeines über Format Provider in .NET

Wenn Sie einen Zahlenwert in eine Zeichenkette umwandeln möchten, dann machen Sie das in der Regel mit der ToString-Funktion, da sie Ihnen die flexibelste Steuerung der Umwandlung bietet. Für numerische Werte stehen Ihnen folgende Möglichkeiten zur Verfügung:

- `DoubleVariable.ToString()`: Der Wert der Zeichenkette wird mit den aktuellen Kultureinstellungen ausgegeben, die von Ihren Betriebssystemeinstellungen abhängig sind.
- `DoubleVariable.ToString("formatzeichenfolge")`: Die Formatzeichenfolge bestimmt, wie der Zahlenwert in eine Zeichenkette umgewandelt werden soll.
- `DoubleVariable.ToString(IFormatProvider)`: Ein so genannter Format Provider (etwa: Formatierungsanbieter) bestimmt, wie die Zeichenfolge formatiert werden soll.

- `DoubleVariable.ToString("formatzeichenfolge", IFormatProvider)`: Eine Formatzeichenfolge bestimmt, wie der Zahlenwert in eine Zeichenkette umgewandelt werden soll; der zusätzlich angegebene Format Provider regelt nähere Konventionen.

Das Gleiche gilt bei der Ausgabe von Datums- bzw. Zeitwerten, wenn also der Datentyp `Date` in eine Zeichenkette umgewandelt werden soll.

Von der ersten Möglichkeit haben Sie sicherlich schon oft Gebrauch gemacht. Wichtig zu wissen: Die kulturabhängigen Besonderheiten werden bei der Umwandlung berücksichtigt. Wenn Sie auf einem deutschen .NET-System den Code

```
Module FormatDemos
```

```
    Sub Main()
```

```
        Dim locDouble As Double = 1234.56789
```

```
        Dim locDate As Date = #12/24/2003 10:33:00 PM#
```

```
        Console.WriteLine(locDouble.ToString())
```

```
        Console.WriteLine(locDate.ToString())
```

```
        Console.ReadLine()
```

```
    End Sub
```

```
End Module
```

ablaufen lassen, erhalten Sie die folgende Ausgabe:

```
1234,56789
```

```
24.12.2003 22:33:00
```

Wenn Sie das Programm kompilieren und auf einem englischem oder amerikanischen Computer¹ laufen lassen (und das, ohne es neu zu kompilieren), sieht die Ausgabe schon ganz anders aus:

```
1234.56789
```

```
12/24/2003 10:33:00 PM
```

Der Hintergrund: Auch wenn Sie keinen zusätzlichen Parameter bei der Verwendung von `ToString` angegeben haben, muss es irgendein Regelwerk in .NET geben, das bestimmt, wann ein Dezimalkomma wirklich ein Komma und wann ein Punkt ist, so wie auf englischen oder amerikanischen Systemen. Diese Bestimmungen werden von den Format Providern geregelt. Sie können auch auf einem deutschen System so tun, als würden Sie englisch formatieren.

Kulturabhängige Formatierungen mit `CultureInfo`

Um länderspezifische Formatierungen bei der Umwandlung vorzunehmen, verwenden Sie einen Format Provider namens `CultureInfo`. Auf diese Klasse können Sie zugreifen, wenn Sie den Namensbereich `System.Globalization` in Ihr Programm eingebunden haben. Verändern Sie das vorherige Programm wie folgt (Änderungen sind fett hervorgehoben):

¹ Es gilt das zuvor Gesagte: Die Ländereinstellungen des Betriebssystems sind dafür verantwortlich.

```

Sub Main()

    Dim locDate As Date = #12/24/2003 10:33:00 PM#
    Dim locDouble As Double = 1234.56789
    Dim locCultureInfo As New CultureInfo("en-US")

    Console.WriteLine(locDouble.ToString(locCultureInfo))
    Console.WriteLine(locDate.ToString(locCultureInfo))
    Console.ReadLine()

End Sub

```

End Module

Wenn Sie dieses Programm starten, werden Sie sehen, dass die Ausgabe exakt dem entspricht, was auf dem US-Computer bei der Verwendung des vorherigen Programms ausgegeben wurde.

Wenn Sie keinen Format Provider bei der ToString-Methode eines primitiven Datentyps angeben, verwendet .NET automatisch den, der der eingestellten Kultur entspricht. Sie können den Format Provider der aktuellen Kultureinstellung mit der statischen Funktion CurrentCulture der CultureInfo-Klasse ermitteln. Wenn Sie die beiden Zeilen

```

Console.WriteLine(locDouble.ToString(locCultureInfo))
Console.WriteLine(locDate.ToString(locCultureInfo))

```

gegen die Zeilen

```

Console.WriteLine(locDouble.ToString(CultureInfo.CurrentCulture))
Console.WriteLine(locDate.ToString(CultureInfo.CurrentCulture))

```

austauschen und wieder einmal auf einem deutschen und einmal auf einem amerikanischen System laufen lassen, bekommen Sie exakt das Ergebnis, das Sie auch beim Beispiel beobachten konnten, bei dem *ToString* gar kein Parameter übergeben wurde.

Wie Sie im vorherigen Beispiel sehen konnten, erstellen Sie eine neue CultureInfo-Klasseninstanz, indem Sie eine Buchstabenkombination angeben, die die entsprechende Kultur bezeichnet. Die folgende Tabelle zeigt Ihnen die wichtigsten Einstellungen. Eine vollständige Tabelle können Sie aus der Visual Studio-Online-Hilfe erhalten.

Kulturkürzel	Kulturcode	Sprache-Land/Region
""(leere Zeichenfolge)	0x007F	Invariante Kultur
Nl	0x0013	Niederländisch
nl-BE	0x0813	Niederländisch – Belgien
nl-NL	0x0413	Niederländisch – Niederlande
En	0x0009	Englisch
en-AU	0x0C09	Englisch – Australien
en-CB	0x2409	Englisch – Karibik
en-IE	0x1809	Englisch – Irland ▶

Kulturkürzel	Kulturcode	Sprache-Land/Region
en-GB	0x0809	Englisch – Großbritannien
en-US	0x0409	Englisch – USA
Fr	0x000C	Französisch
fr-BE	0x080C	Französisch – Belgien
fr-CA	0x0C0C	Französisch – Kanada
fr-FR	0x040C	Französisch – Frankreich
fr-LU	0x140C	Französisch – Luxemburg
fr-MC	0x180C	Französisch – Monaco
fr-CH	0x100C	Französisch – Schweiz
de	0x0007	Deutsch
de-AT	0x0C07	Deutsch – Österreich
de-DE	0x0407	Deutsch – Deutschland
de-LI	0x1407	Deutsch – Liechtenstein
de-LU	0x1007	Deutsch – Luxemburg
de-CH	0x0807	Deutsch – Schweiz
it	0x0010	Italienisch
it-IT	0x0410	Italienisch – Italien
it-CH	0x0810	Italienisch – Schweiz
es	0x000A	Spanisch
es-AR	0x2C0A	Spanisch – Argentinien
es-PR	0x500A	Spanisch – Puerto Rico
es-ES	0x0C0A	Spanisch – Spanien
es-VE	0x200A	Spanisch – Venezuela

Tabelle 17.1: Diese Einstellungen verwenden Sie für *CulturInfo*-Objekte

Vermeiden von kulturabhängigen Programmfehlern

Auf Programmentwicklungen hat das unter Umständen Auswirkungen, nämlich dann, wenn Sie numerische Daten oder Zeitdaten in Textdateien speichern und diese kulturübergreifend ausgetauscht werden müssen. Und: Das Definieren von konstanten Werten sollte im Programm nur direkt mit Literalen und nie durch Zeichenketten und entsprechende Konvertierungsfunktionen erfolgen. Dazu ein Beispiel:

Die Codezeilen

```
Dim locDoubleTest As Double = CDb1("1.234")
Console.WriteLine(locDoubleTest)
```

ergeben auf einem deutschen System den Wert 1234; auf einem englischsprachigen System jedoch den Wert 1,234 (aber 1.234 angezeigt), da hier der Punkt nicht als Tausendergruppierung, sondern eben als Dezimalkomma (*Decimal Point*) angesehen wird. Mehr Überlegungen zu diesem Thema finden Sie in auch im vorherigen Kapitel. Das dort Gesagte gilt gleichermaßen auch für Datumswerte. Die Umwandlung aus einer Zeichenfolge wirkt dabei noch konstruiert, aber denken Sie daran, wie oft Eingaben aus einer TextBox mit `TextBox.Text` (eine Methode, die einen String zurückgibt) geparkt werden müssen, auch und gerade für numerische Werte.

Wenn Sie Formatierungen für das Serialisieren (Speichern) vornehmen müssen, verwenden Sie dazu am besten ein so genanntes *invariantes* `CultureInfo-Objekt`, das Sie mit folgender Anweisung ermitteln können:

```
Dim locCultureInfo As CultureInfo = CultureInfo.InvariantCulture
```

Wenn Sie alle Konvertierungen innerhalb Ihres Programms (also sowohl das Parsen eines Strings, um ihn in einen entsprechenden Datentyp umzuwandeln, als auch das Konvertieren eines Datentyps in einen String) damit durchführen, sind Sie auf der sicheren Seite.

Formatierung durch Formatzeichenfolgen

Sowohl die `ToString`-Methode als auch die `Parse`- bzw. die `ParseExact`-Methode unterstützen neben dem Einsatz von `Format` Providern auch die Formatierung durch direkte Mustervorlagen, die durch Strings oder `String`-Arrays übergeben werden. Sie haben schon an einigen Beispielen gesehen, dass bestimmte Buchstaben, zu Gruppen zusammengeführt, bestimmte Formatierungen eines Datenblocks (Tage, Monate oder Jahre beispielsweise) bewirken. Die folgenden Abschnitte demonstrieren den Einsatz von Formatzeichenfolgen.

Formatierung von numerischen Ausdrücken durch Formatzeichenfolgen

BEGLEITDATEIEN: Sie finden im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap17\FormatProvider - Num\` die Beispiele, die in diesem Abschnitt behandelt werden.

Wenn Sie die Codezeilen,

```
Dim locDouble As Double
locDouble = Math.PI + 10000 * Math.PI
Console.WriteLine("locDouble hat den Wert:" + locDouble.ToString())
```

ausführen, erhalten Sie die Ausgabe

```
locDouble hat den Wert: 31419,0681285515
```

In vielen Fällen ist es aber erwünscht, beispielsweise den Tausendergruppierungspunkt ebenfalls anzeigen zu lassen oder nur eine bestimmte Anzahl von Nachkommastellen auszugeben. Dann können Sie die Ausgabe durch Formatzeichenfolgen reglementieren.

Wenn Sie beispielsweise die Zeile

```
Console.WriteLine("locDouble hat den Wert:" + locDouble.ToString())
```

des obigen Beispiels durch die folgende ersetzen,

```
Console.WriteLine("locDouble hat den Wert: " + locDouble.ToString("#,###.00"))
```

sieht die Ausgabe schon sehr viel ordentlicher aus:

```
locDouble hat den Wert: 31.419,07
```

In diesem Fall ist der der ToString-Methode übergebene Parameter als Regelvorgabe für die Formatierung übergeben worden. Jedes »#«-Zeichen stellt dabei eine mögliche Ziffer dar, jede 0 eine Muss-Ziffer. Das bedeutet: Die Zahl 304 würde keine zusätzliche führende 0 bei der Ausgabe bekommen, da die Formatzeichenfolge zwar vier Zeichen (»#,###«) vorgibt, durch die Verwendung des »#«-Zeichens aber keine zwingende Verwendung aller Ziffern vorgeschrieben wird.

Anders sieht es bei dem gleichen Wert und seinen Nachkommastellen nach der Formatierung aus: Da hier »0« und nicht »#« angegeben wird, lauten die Nachkommastellen nach der Formatierung »00«, obwohl zur Wertdarstellung eigentlich keine Nachkommastellen nötig wären.

Die folgende Tabelle zeigt, welche Formatzeichen Sie bei der Zusammenstellung von Formatzeichenfolgen für numerische Werte verwenden können:

Formatzeichen	Name	Beschreibung
0	0-Platzhalter	<p>Gibt es an der Stelle des 0-Platzhalters eine korrelierende Ziffer im aufzubereitenden Wert, dann wird die Ziffer des Wertes an der Stelle platziert, die der 0-Platzhalter durch seine Position vorgibt.</p> <p>Wenn der zu formatierende Wert die vorgegebene Ziffernposition nicht hergibt (bei »000,00« als Vorgabe gäbe es für den Wert 34,1 nicht alle vorgegebenen Position), wird eine 0 angezeigt.</p> <p>Die Positionen der »0«, die am weitesten links vor dem Dezimaltrennzeichen steht, und der »0«, die am weitesten rechts hinter dem Dezimaltrennzeichen steht, bestimmen also den Bereich der Ziffern, die immer in der Ergebniszeichenfolge enthalten sind.</p> <p>Falls im zu formatierenden Wert mehr Nachkommastellen vorhanden sind, als Nullen Positionen vorgeben, wird auf die entsprechende Anzahl auf Stellen gerundet.</p>
#	Ziffernplatzhalter	<p>Verfügt der zu formatierende Wert über eine Ziffer an der Stelle, an der »#« in der Formatzeichenfolge steht, wird diese Ziffer in der Ergebniszeichenfolge angezeigt, anderenfalls nicht.</p> <p>Beachten Sie, dass dieses Formatzeichen nie die Anzeige von 0 bewirkt, wenn es sich nicht um eine signifikante Ziffer handelt, selbst wenn 0 die einzige Ziffer in der Zeichenfolge ist. 0 wird jedoch angezeigt, wenn es sich um eine signifikante Ziffer in der angezeigten Zahl handelt (bei 0304,030 beispielsweise sind die äußeren Nullen nicht signifikant, d. h. Weglassen verändert den Wert nicht).</p>
.	Dezimaltrennzeichen	<p>Das erste ».«-Zeichen in der Formatzeichenfolge bestimmt die Position des Dezimaltrennzeichens im formatierten Wert. Weitere ».«-Zeichen werden ignoriert. Das für das Dezimaltrennzeichen verwendete Zeichen wird durch die <i>NumberDecimalSeparator</i>-Eigenschaft der <i>NumberFormatInfo</i> bestimmt, die die Formatierung steuert, allerdings nur für die Ausgabe, nie für die Vorgabe! WICHTIG: Verwechseln Sie ».« und ».« nicht bei der Erstellung der Formatzeichenfolge. Die amerikanisch/englische Formatvorgabe ist hier maßgeblich – beispielsweise »1,000,000.23« für Einemillionkomma-zweidrei. ▶</p>

Formatzeichen	Name	Beschreibung
,	Tausendertrennzeichen und Zahlenskalierung	Das »,-«-Zeichen erfüllt zwei Aufgaben. Erstens: Wenn die Formatzeichenfolge ein »,-«-Zeichen zwischen zwei Ziffernplatzhaltern enthält (»0« oder »#«) und einer der Platzhalter links neben dem Dezimaltrennzeichen steht, werden in der Ausgabe Tausendertrennzeichen zwischen jeder Gruppe von drei Ziffern links neben dem Dezimaltrennzeichen eingefügt. Das in der Ergebniszeichenfolge als Dezimaltrennzeichen verwendete Zeichen wird durch die <i>NumberGroupSeparator</i> -Eigenschaft der aktuellen <i>NumberFormatInfo</i> bestimmt, die die Formatierung steuert. Zweitens: Wenn die Formatzeichenfolge ein oder mehr »,-«-Zeichen direkt links neben dem Dezimaltrennzeichen enthält, wird die Zahl durch die Anzahl der »,-«-Zeichen multipliziert mit 1000 dividiert, bevor sie formatiert wird. Die Formatzeichenfolge »0,-« stellt 100 Millionen z. B. als 100 dar. Verwenden Sie das »,-«-Zeichen, um anzugeben, dass die Skalierung keine Tausendertrennzeichen in der formatierten Zahl enthält. Um also eine Zahl um 1 Million zu skalieren und Tausendertrennzeichen einzufügen, verwenden Sie die Formatzeichenfolge »#,##0,-«. WICHTIG: Verwechseln Sie »,-« und ».-« nicht bei der Erstellung der Formatzeichenfolge. Die amerikanisch/englische Formatvorgabe ist hier maßgeblich – beispielsweise »1,000,000.23« für <i>Einemillionkommazweidrei</i> .
%	Prozentplatzhalter	Enthält eine Formatzeichenfolge ein »%«-Zeichen, wird die Zahl vor dem Formatieren mit 100 multipliziert. Das entsprechende Symbol wird in der Zahl an der Stelle eingefügt, an der »%« in der Formatzeichenfolge steht. Das verwendete Prozentzeichen ist von der aktuellen <i>NumberFormatInfo</i> -Klasse abhängig.
E0 E+0 E-0 e0 e+0 e-0	Wissenschaftliche Notation	Enthält die Formatzeichenfolge die Zeichenfolgen »E«, »E+«, »E-«, »e«, »e+« oder »e-« und folgt direkt danach mindestens ein »0«-Zeichen, wird die Zahl mit der wissenschaftlichen Notation formatiert und ein »E« bzw. »e« zwischen der Zahl und dem Exponenten eingefügt. Die Anzahl der »0«-Zeichen nach dem entsprechenden Formatzeichen für die wissenschaftliche Notation bestimmt die Mindestanzahl von Ziffern, die für den Exponenten ausgegeben werden. Das »E+«-Format und das »e+«-Format geben an, dass immer ein Vorzeichen (Plus oder Minus) vor dem Exponenten steht. Die Formate »E«, »E-«, »e« oder »e-« geben an, dass nur vor negativen Exponenten ein Vorzeichen steht.
'ABC' "ABC"	Zeichenfolgenliteral	Zeichen, die in einfachen bzw. doppelten Anführungszeichen stehen, werden direkt in die Ergebniszeichenfolge kopiert, ohne die Formatierung zu beeinflussen.
;	Abschnitttrennzeichen	Mit dem »;-«-Zeichen werden Abschnitte für positive und negative Zahlen sowie Nullen in der Formatzeichenfolge voneinander getrennt. Lesen Sie dazu auch die Anmerkung am Ende der Tabelle.
Sonstige	Alle anderen Zeichen	Alle anderen Zeichen werden als Literale an der angegebenen Position in die Ergebniszeichenfolge kopiert.

Tabelle 17.2: Für numerische Formatzeichenfolgen verwenden Sie diese Formatzeichen

HINWEIS: Bitte beachten Sie, dass Sie für negative und positive Werte sowie für den Wert 0 jeweils eine individuelle Zeichenfolge bestimmen können, die durch das Semikolon getrennt werden. Das folgende Beispiel verdeutlicht ihre Anwendung:

```

Dim locDouble As Double
Dim locFormat As String = "#,###.00;-#,###.0000;+-0.00000"
locDouble = Math.PI + 10000 * Math.PI
Console.WriteLine("locDouble hat den Wert: " + locDouble.ToString(locFormat))
locDouble *= -1
Console.WriteLine("locDouble hat den Wert: " + locDouble.ToString(locFormat))
locDouble = 0
Console.WriteLine("locDouble hat den Wert: " + locDouble.ToString(locFormat))

```

Dieser Codeausschnitt würde das folgende Ergebnis auf dem Bildschirm produzieren:

```

locDouble hat den Wert: 31.419,07
locDouble hat den Wert: -31.419,0681
locDouble hat den Wert: +-0,00000

```

Formatierung von numerischen Ausdrücken durch vereinfachte Formatzeichenfolgen

Für den einfachen Einsatz gibt es in .NET einige vereinfachte Formatzeichenfolgen, die in der Regel nur aus einem einzigen Zeichen bestehen und einen Typ von Formatierung bezeichnen. Anstatt z.B. eine Formatzeichenfolge zu erstellen, die eine Währungsformatierung vorgibt, können Sie einfach das »C«-Zeichen als Formatzeichenfolge verwenden. In Kombination mit dem entsprechenden *CultureInfo*-Format-Provider brauchen Sie sich obendrein noch nicht einmal um das Finden des entsprechenden Währungssymbols zu kümmern:

```

Dim locDouble As Double

locDouble = 12234.346
Console.WriteLine("Sie bekommen {0} aus einem Lottogewinn.", locDouble.ToString("$#,###.00"))
Console.WriteLine("Sie bekommen {0} aus einem Lottogewinn.", locDouble.ToString("C", New CultureInfo("en-US")))

```

Beide *Console.WriteLine*-Anweisungen zeigen in diesem Beispiel das exakt gleiche Ergebnis im Konsolenfenster an, nämlich:²

```

Sie bekommen $12.234,35 aus einem Lottogewinn.
Sie bekommen $12,234.35 aus einem Lottogewinn.

```

Die folgende Tabelle zeigt Ihnen, welche Kurzformen für Formatzeichenfolgen .NET für die Formatierung von numerischen Werten kennt. Bitte beachten Sie, dass, wie im Beispiel gezeigt, die Resultate unter Umständen kulturabhängig sind und sich deswegen mit einem *CulturInfo*-Objekt entsprechend steuern lassen.

Formatzeichen	Beschreibung
c, C	Währungsformat.
d, D	Dezimales Format. ▶

² Ich habe bei Währungsangaben bewusst auf die Euro-Währung verzichtet, da Konsolenanwendungen das Euro-Zeichen nicht ohne weiteres darstellen können.

Formatzeichen	Beschreibung
e, E	Wissenschaftliches Format (Exponentialformat).
f, F	Festkommaformat.
g, G	Allgemeines Format.
n, N	Zahlenformat.
r, R	Schleifenformat, das sicherstellt, dass in Zeichenfolgen konvertierte Zahlen denselben Wert haben, wenn sie wieder in Zahlen konvertiert werden.
x, X	Hexadezimalformat.

Tabelle 17.3: Für vereinfachte numerische Formatzeichenfolgen verwenden Sie diese Formatzeichen

Formatierung von Datums- und Zeitausdrücken durch Formatzeichenfolgen

BEGLEITDATEIEN: Sie finden im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap17\FormatProvider - Date\` die Beispiele, die in diesem Abschnitt behandelt werden.

Was für numerische Datentypen gilt, ist auch auf den Datentyp `Date` anwendbar: Es gibt eine umfangreiche Unterstützung durch bestimmte Formatzeichenfolgen, die bestimmen, wie ein Datumswert in eine Zeichenkette umgewandelt werden kann.

Wenn Sie die Codezeilen,

```
Dim locDate As Date
locDate = #12/24/2003 10:33:00 PM#
Console.WriteLine("locDate hat den Wert: " + locDate.ToString())
```

ausführen, erhalten Sie die Ausgabe

```
locDate hat den Wert: 24.12.2003 22:33:00
```

In den meisten Fällen benötigen Sie aber keine kombinierte Ausgabe des Zeitwertes von Uhrzeit und Datum sondern nur eine von beiden. Außerdem verlangen viele Anwendungen, das Datum etwas ausführlicher darzustellen – beispielsweise indem die ersten Buchstaben des Monats dargestellt werden oder der Wochentag des Datums mit ausgegeben wird.

Wenn Sie beispielsweise die Zeile

```
Console.WriteLine("locDate hat den Wert: " + locDate.ToString())
```

des obigen Beispiels durch die folgende ersetzen,

```
Console.WriteLine("locDate hat den Wert: " + locDate.ToString("dddd, dd. MMM yyyy - HH:mm"))
```

erkennen Sie, dass Sie durch den Einsatz einer Datums-Formatzeichenfolgen sehr viel mehr Einfluss auf die Art der Umwandlung des Datumswertes in eine Zeichenkette genommen haben, denn die Ausgabe sieht nun folgendermaßen aus:

```
locDate hat den Wert: Mittwoch, 24. Dez 2003 - 22:33
```

In diesem Fall haben Sie der `ToString`-Methode eine Zeichenkette mit Formatanweisungen für die Formatierung übergeben. Jedes der verwendeten Zeichen der Formatzeichenfolge spiegelt dabei eine Datengruppe innerhalb eines `Date`-Datentyps wider.

Die folgende Tabelle zeigt, welche Formatzeichen Sie bei der Zusammenstellung von Formatzeichenfolgen für numerische Werte verwenden können.

WICHTIG: Einige Formatzeichen der folgenden Tabelle können auch den Kurzformatzeichen der übernächsten Tabelle entsprechen. Achten Sie deswegen darauf, dass Sie die folgenden Zeichen im beschriebenen Kontext nicht alleine verwenden. Und: Je nach zusätzlich verwendeter `CultureInfo` produzieren die verwendeten Formatzeichenkombinationen unterschiedliche Resultate.

Formatzeichen	Beschreibung
d	Zeigt den aktuellen Tag des Monats als Zahl zwischen 1 und 31 an. Wenn die Nummer des Tages nur einstellig ist, wird diese nicht mit einer führenden Null versehen.
dd	Zeigt den aktuellen Tag des Monats als Zahl zwischen 1 und 31 an. Wenn die Nummer des Tages nur einstellig ist, wird ihr eine 0 vorangestellt.
ddd	Zeigt den abgekürzten Namen des Tages für den angegebenen <i>Date</i> -Wert an.
dddd	Zeigt den vollständigen Namen des Tages für den angegebenen <i>Date</i> -Wert an.
f	Zeigt die Bruchteile von Sekunden als eine Ziffer an.
ff	Zeigt die Bruchteile von Sekunden als zwei Ziffern an.
fff	Zeigt die Bruchteile von Sekunden als drei Ziffern an.
ffff	Zeigt die Bruchteile von Sekunden als vier Ziffern an.
Ffff	Zeigt die Bruchteile von Sekunden als fünf Ziffern an.
ffffff	Zeigt die Bruchteile von Sekunden als sechs Ziffern an.
fffffff	Zeigt die Bruchteile von Sekunden als sieben Ziffern an.
g oder gg	Zeigt den Zeitraum für den angegebenen <i>Date</i> -Wert an. Wenn Sie mit einer deutschen Kultureinstellung (durch <i>CultureInfo</i> bestimmt) arbeiten, ist das beispielsweise der Vermerk »n. Christ.«.
H	Zeigt die Stunde des angegebenen <i>Date</i> -wertes im Bereich 1–12 an. Die Stunde stellt die ganzen Stunden dar, die seit Mitternacht (als 12 dargestellt) oder Mittag (ebenfalls als 12 dargestellt) vergangen sind. Wenn dieses Format einzeln verwendet wird, können die jeweils gleichen Stunden vor und nach Mittag nicht unterschieden werden. Wenn die Stundenzahl nur einstellig ist, wird diese auch als einzelne Ziffer angezeigt. Wichtig: Für das 24-Stunden-Format wählen Sie das große »H«.
Hh	Zeigt die Stunde des angegebenen <i>Date</i> -wertes im Bereich 1–12 an. Die Stunde stellt die ganzen Stunden dar, die seit Mitternacht (als 12 dargestellt) oder Mittag (ebenfalls als 12 dargestellt) vergangen sind. Wenn dieses Format einzeln verwendet wird, können die jeweils gleichen Stunden vor und nach Mittag nicht unterschieden werden. Wenn die Stundenzahl nur einstellig ist, wird eine führende Null vorangestellt. Wichtig: Für das 24-Stunden-Format wählen Sie das große »H«.
H	Zeigt die Stunde des angegebenen <i>Date</i> -wertes im Bereich 0–23 an. Die Stunde stellt die seit Mitternacht (als 0 angezeigt) vergangenen Stunden dar. Wenn die Stundenzahl nur einstellig ist, wird sie auch nur als einzelne Ziffer angezeigt. ►

Formatzeichen	Beschreibung
HH	Zeigt die Stunde des angegebenen <i>Date</i> -Wertes im Bereich 0–23 an. Die Stunde stellt die seit Mitternacht (als 0 angezeigt) vergangenen Stunden dar. Wenn die Stundenzahl einstellig ist, wird dieser Ziffer eine führende 0 vorangestellt.
m	Zeigt die Minute des angegebenen <i>Date</i> -Wertes im Bereich 0–59 an. Die Minute stellt die seit der letzten Stunde vergangenen ganzen Minuten dar. Wenn die Minute nur einstellig ist, wird diese auch nur als einzelne Ziffer angezeigt.
mm	Zeigt die Minute des angegebenen <i>Date</i> -Wertes im Bereich 0–59 an. Die Minute stellt die seit der letzten Stunde vergangenen ganzen Minuten dar. Wenn die Minute nur eine einzelne Ziffer ist (0–9), wird dieser Ziffer bei der Formatierung eine 0 (01–09) vorangestellt.
M	Zeigt den Monat als Zahl zwischen 1 und 12 an. Die Ausgabe erfolgt bei einstelligen Werten ohne Führungsnull. Wichtig: Denken Sie daran, hier Großbuchstaben zu verwenden, da Sie sonst eine Minutenausgabe erwirken.
MM	Zeigt den Monat als Zahl zwischen 1 und 12 an. Die Ausgabe erfolgt bei einstelligen Werten mit Führungsnull. Wichtig: Denken Sie daran, hier Großbuchstaben zu verwenden, da Sie sonst eine Minutenausgabe erwirken.
MMM	Zeigt den abgekürzten Namen des Monats für den angegebenen <i>Date</i> -Wert an. Wichtig: Denken Sie daran, hier Großbuchstaben zu verwenden, da Sie sonst eine Minutenausgabe erwirken.
MMMM	Zeigt den vollständigen Namen des Monats für den angegebenen <i>Date</i> -Wert an. Wichtig: Denken Sie daran, hier Großbuchstaben zu verwenden, da Sie sonst eine Minutenausgabe erwirken.
s	Zeigt die Sekunden im Bereich 0–59 an. Die Sekunde stellt die seit der letzten Minute vergangenen ganzen Sekunden dar. Die Ausgabe erfolgt bei einstelligen Werten ohne Führungsnull.
ss	Zeigt die Sekunden im Bereich 0–59 an. Die Sekunde stellt die seit der letzten Minute vergangenen ganzen Sekunden dar. Die Ausgabe erfolgt bei einstelligen Werten mit Führungsnull.
t	Zeigt das erste Zeichen des A.M./P.M.-Bezeichners für den angegebenen <i>Date</i> -Wert an. Wichtig: Beim deutschen <i>CultureInfo</i> -Format-Provider (entspricht auf deutschen .NET-Systemen der Standardeinstellung, wenn kein Format Provider angegeben wurde), bleibt dieses Formatzeichen unberücksichtigt – wird also ignoriert.
tt	Zeigt den vollständigen A.M./P.M.-Bezeichner für den angegebenen <i>Date</i> -Wert an. Wichtig: Beim deutschen <i>CultureInfo</i> -Format-Provider (entspricht auf deutschen .NET-Systemen der Standardeinstellung, wenn kein Format Provider angegeben wurde), bleibt dieses Formatzeichen unberücksichtigt – wird also ignoriert.
y	Zeigt das Jahr an. Die ersten beiden Ziffern des Jahres werden ausgelassen. Die Ausgabe erfolgt bei einstelligen Werten ohne Führungsnull.
yy	Zeigt das Jahr an. Die ersten beiden Ziffern des Jahres werden ausgelassen. Die Ausgabe erfolgt bei einstelligen Werten mit Führungsnull.
yyyy	Zeigt das Jahr an. Wenn das Jahr aus weniger als vier Ziffern besteht, werden diesem Nullen vorangestellt, damit es vier Ziffern enthält.
z	Zeigt das Offset der aktuellen Zeitzone des Systems in ganzen Stunden an. Das Offset wird immer mit einem vorangestellten Plus- bzw. Minuszeichen angezeigt (Null wird als »+0« angezeigt), das die Stunden vor (+) GMT (Greenwich Mean Time) bzw. nach (-) GMT angibt. Der Wertebereich liegt zwischen –12 und +13 Stunden. Ist das Offset einstellig, erfolgt die Anzeige ebenfalls nur einstellig; es werden keine führenden Nullen vorangestellt. Die Einstellung für die Zeitzone wird als +X oder –X angegeben, wobei X der Offset zur GMT in Stunden ist. Das angezeigte Offset kann durch die Sommer-/Winterzeitumstellungen beeinflusst werden.
zz	Wie vorher; das Offset wird bei einstelligen Werten jedoch mit führender Null angezeigt. ►

Formatzeichen	Beschreibung
zzz	Wie vorher; das Offset wird aber in Stunden und Minuten angezeigt.
:	Trennzeichen für Zeitangaben.
/	Trennzeichen für Datumsangaben. WICHTIG: Wenn Sie für das Parsen eines Datumswertes einen Schrägstrich verlangen, müssen Sie »\« verwenden, um den Schrägstrich als Gruppentrennzeichen zu bestimmen.
"	Zeichenfolge in Anführungszeichen. Zeigt den literalen Wert einer Zeichenfolge zwischen zwei Anführungszeichen an, denen ein Escapezeichen (/) vorangestellt ist.
'	Zeichenfolge in Anführungszeichen. Zeigt den literalen Wert einer Zeichenfolge zwischen zwei »'«-Zeichen an.
Jedes andere Zeichen	Andere Zeichen werden als Literale direkt in die Ergebniszeichenfolge geschrieben.

Tabelle 17.4: Für Formatzeichenfolgen zur Aufbereitung von Datums- und Zeitwerten verwenden Sie diese Formatzeichen

Die folgenden Beispiele sollen den Umgang mit den Formatzeichenfolgen verdeutlichen:

Module FormatDemosFormatzeichenfolgen

Sub Main()

Dim locDate As Date = #12/24/2005 10:32:22 PM#

Dim locFormat As String

'Normale Datumsausgabe; große Ms ergeben den Monat.

locFormat = "dd.MM.yyyy"

```
Console.WriteLine("{0} : {1}", _
    locFormat, _
    locDate.ToString(locFormat, CultureInfo.CurrentCulture))
```

'Falsche Ausgabe; anstelle des Monats werden Minuten ausgegeben.

locFormat = "dd.mm.yyyy"

```
Console.WriteLine("{0} : {1}", _
    locFormat, _
    locDate.ToString(locFormat, CultureInfo.CurrentCulture))
```

'Komplette Ausgabe, ausführlicher geht's nicht.

locFormat = "dddd, dd.MMMM.yyyy - HH:mm:ss:ffffff "(Offset:" zzz)"

```
Console.WriteLine("{0} : {1}", _
    locFormat, _
    locDate.ToString(locFormat, CultureInfo.CurrentCulture))
```

'Falsche Ausgabe; der Text "Uhrzeit" steht nicht in Anführungszeichen.

locFormat = "Uhrzeit: HH:mm:ss"

```
Console.WriteLine("{0} : {1}", _
    locFormat, _
    locDate.ToString(locFormat, CultureInfo.CurrentCulture))
```

'So geht es richtig:

locFormat = ""Uhrzeit:" HH:mm:ss"

```
Console.WriteLine("{0} : {1}", _
    locFormat, _
    locDate.ToString(locFormat, CultureInfo.CurrentCulture))
```

```

'PM-Anzeige funktioniert nicht bei deutscher...
locFormat = ""Uhrzeit:"" hh:mm:ss tt"
Console.WriteLine("{0} : {1}", _
    locFormat, _
    locDate.ToString(locFormat, New CultureInfo("de-DE")))

'...aber beispielsweise bei amerikanischer Kultureinstellung
locFormat = ""Uhrzeit:"" hh:mm:ss tt"
Console.WriteLine("{0} : {1}", _
    locFormat, _
    locDate.ToString(locFormat, New CultureInfo("en-US")))

'Englisches Datumsformat trotz deutscher Kultureinstellung
locFormat = ""Date:"" MM/dd/yyyy"
Console.WriteLine("{0} : {1}", _
    locFormat, _
    locDate.ToString(locFormat, New CultureInfo("de-DE")))

'Backslash davor nicht vergessen, sonst:
locFormat = ""Date:"" MM/dd/yyyy"
Console.WriteLine("{0} : {1}", _
    locFormat, _
    locDate.ToString(locFormat, New CultureInfo("de-DE")))

'Aber nur so mit englischen Texten:
locFormat = ""Date:"" MMMM, "the" dd. yyyy"
Console.WriteLine("{0} : {1}", _
    locFormat, _
    locDate.ToString(locFormat, New CultureInfo("en-US")))

    Console.ReadLine()
End Sub
End Module

```

Lassen Sie dieses Programm laufen, ergibt sich folgende Ausgabe im Konsolenfenster:

```

'dd.MM.yyyy' : 24.12.2005
'dd.mm.yyyy' : 24.32.2005
'dddd, dd.MMMM.yyyy - HH:mm:ss:ffffff "(Offset:" zzz)' : Samstag, 24.Dezember.2005 - 22:32:22:0000000
(Offset: +01:00)
'Uhrzeit: HH:mm:ss' : U10r+1ei: 22:32:22
'"Uhrzeit:" HH:mm:ss' : Uhrzeit: 22:32:22
'"Uhrzeit:" hh:mm:ss tt' : Uhrzeit: 10:32:22
'"Uhrzeit:" hh:mm:ss tt' : Uhrzeit: 10:32:22 PM
'"Date:" MM/dd/yyyy' : Date: 12/24/2005
'"Date:" MM/dd/yyyy' : Date: 12.24.2005
'"Date:" MMMM, "the" dd. yyyy' : Date: December, the 24. 2005

```

Formatierung von Zeitausdrücken durch vereinfachte Formatzeichenfolgen

Für die bequeme Realisierung von Zeit- und Datumsformatierungen gibt es in .NET einige vereinfachte Formatzeichenfolgen, die in der Regel nur aus einem einzigen Zeichen bestehen und einen Formatierungsstil bezeichnen. Anstatt beispielsweise gezielt die Formatzeichenkombinationen zusammenzustellen, die ein Datum in Langform formatieren, reicht das Zurückgreifen auf ein bestimmtes vereinfachtes Formatzeichen. In Kombination mit dem entsprechenden *CultureInfo*-Format-Provider brauchen Sie sich obendrein noch nicht einmal um die in der Kultur übliche Darstellung zu kümmern:

```
Sub main()  
    Dim locDate As Date = #12/24/2005 10:32:22 PM#  
    Dim locFormat As String  
  
    locFormat = "dddd, dd. MMMM yyyy"  
    Console.WriteLine("Datumsformatierung mit Formatzeichen:" + locDate.ToString(locFormat))  
    Console.WriteLine("...und mit vereinfachten Formatzeichen:" + locDate.ToString("D"))  
    Console.ReadLine()  
End Sub
```

Die folgende Tabelle zeigt, welche vereinfachten Formatzeichenfolgen Sie verwenden können, um Datums- und Zeitformatierungen durchzuführen. Viele der vereinfachten Formatzeichen haben äquivalente Formatzeichenfolgen, die mit entsprechenden (in der Tabelle angegebenen) Eigenschaften mithilfe eines *DateTimeFormatInfo*-Objektes ermittelt werden können.

Formatzeichen	Zugeordnete Eigenschaft/Beschreibung
d	Kurzform des Datums. Entspricht der Formatzeichenfolge, die die Eigenschaft <i>ShortDatePattern</i> zurückliefert.
D	Langform des Datums. Entspricht der Formatzeichenfolge, die die Eigenschaft <i>LongDatePattern</i> zurückliefert.
f	Vollständiges Datum und Uhrzeit (Langform des Datums und 24-Stunden-Zeitformat).
F	Langform des Datums und der Uhrzeit. Entspricht der Formatzeichenfolge, die die Eigenschaft <i>FullDateTimePattern</i> zurückliefert.
g	Allgemein (Kurzform des Datums sowie 24-Stunden-Zeitformat).
G	Allgemein (Kurzform des Datums und Langform der Zeit).
m, M	Ergibt den Tag und den ausgeschriebenen Monatsnamen (beispielsweise 24. Dezember). Entspricht der Formatzeichenfolge, die die Eigenschaft <i>MonthDayPattern</i> zurückliefert.
r, R	Ergibt Datum und Zeit in der so genannten RFC1123-Norm. Dabei hat die Zeiteingabe das Format beispielsweise von » Sat, 24 Dec 2005 22:32:22 GMT«. Entspricht der Formatzeichenfolge, die die Eigenschaft <i>RFC1123Pattern</i> zurückliefert. Diese Form wird bei der Kommunikation über das Internet verwendet, beispielsweise bei der <i>Header</i> -Dokumentierung von Mail-Dateien.
s	Ergibt ein auf der Grundlage von ISO 8601 unter Verwendung der Ortszeit formatiertes sortierbares Datum (beispielsweise » 2005-12-24T22:32:22«). Entspricht der Formatzeichenfolge, die die Eigenschaft <i>SortableDateTimePattern</i> zurückliefert.
t	Kurzform der Zeit. Entspricht der Formatzeichenfolge, die die Eigenschaft <i>ShortTimePattern</i> zurückliefert.
T	Langform der Zeit. Entspricht der Formatzeichenfolge, die die Eigenschaft <i>LongTimePattern</i> zurückliefert. ►

Formatzeichen	Zugeordnete Eigenschaft/Beschreibung
u	Ergibt ein unter Verwendung des Formats zur Anzeige der koordinierten Weltzeit formatiertes sortierbares Datum (beispielsweise »2005-12-24 22:32:22Z«). Entspricht der Formatzeichenfolge, die die Eigenschaft <i>UniversalSortableDateTimePattern</i> zurückliefert.
U	Vollständiges Datum und Uhrzeit (langes Datumsformat und langes Zeitformat) unter Verwendung der koordinierten Weltzeit.
y, Y	Monat und Jahreszahl (etwa »Dezember 2005«). Entspricht der Formatzeichenfolge, die die Eigenschaft <i>YearMonthPattern</i> zurückliefert.

Tabelle 17.5: Für vereinfachte Formatzeichen zur Aufbereitung von Datums- und Zeitwerten verwenden Sie diese Tabelle

Gezielte Formatierungen mit Format Providern

Das Ziel bei der Implementierung von Format Providern in .NET war es, dem Entwickler ein möglichst universelles Werkzeug zum Aufbereiten von Daten zum Zweck der übersichtlichen Ausgabe auf Bildschirmen oder in Dateien zu bieten. Gleichzeitig sollten kulturabhängige Formatierungseigenarten berücksichtigt werden.

Letzteres haben Sie bereits in Form des *CultureInfo*-Objektes kennen gelernt. Wenn Sie der *ToString*-Funktion eines primitiven Datentyps ein *CultureInfo*-Objekt übergeben, das unter Angabe eines bestimmten Kulturkennzeichens instanziiert wurde, passt sich die Ausgabe an die in der Kultur übliche Datendarstellungsweise an.

Neben dem *CultureInfo*-Objekt kennt das Framework zwei weitere Format Provider, die die Formatierung von numerischen Werten (*NumberFormatInfo*) und Datums- und Zeitwerten (*DateTimeFormatInfo*) genauer spezifizieren. Diese Format Provider arbeiten in Zusammenarbeit mit den vereinfachten Formatzeichen.

Gezielte Formatierungen von Zahlenwerten mit *NumberFormatInfo*

Angenommen, Sie möchten eine Reihe von Zahlen als Währung formatiert untereinander ausgeben. Sie möchten dabei, dass die Ziffern der einzelnen Zahlen zwar als Tausender gruppiert werden (also 1.000.000 und nicht 1000000), aber Sie möchten nicht den Punkt, sondern das Leerzeichen als Gruppierungstrennzeichen verwenden. In diesem Fall verwenden Sie eine *NumberFormatInfo*-Instanz, um die Formatierungsregel genauer zu spezifizieren:

```
Sub main()
    'Den zu verwendenden Wert definieren.
    Dim locDouble As Double = 1234567.23
    'Kultureinstellungen sind englisch/britisch.
    Dim locCultureInfo As New CultureInfo("en-GB")
    'Die Einstellungen, die CultureInfo auf Grund der Kultur schon
    'geleistet hat, über nehmen wir in ein NumberFormatInfo..
    Dim locNumFormatInfo As NumberFormatInfo = locCultureInfo.NumberFormat

    '...dessen anzuwendende Regeln wir nun genauer spezifizieren können:
    locNumFormatInfo.CurrencyGroupSeparator = " "
    Console.WriteLine("Als Währung formatiert: " + locDouble.ToString("C", locNumFormatInfo))
End Sub
```

```

'Auf die normale Fließkommadarstellung hat diese Einstellung keinen Einfluss:
Console.WriteLine("Als Fließkommazahl formatiert: " + locDouble.ToString("n", locNumFormatInfo))

'Jetzt schon!
locNumFormatInfo.NumberGroupSeparator = " "
Console.WriteLine("Als Fließkommazahl formatiert: " + locDouble.ToString("n", locNumFormatInfo))
End Sub

```

Wenn Sie diese kleine Prozedur laufen lassen, produziert sie folgende Ausgabe:

```

Als Wahrung formatiert: £1 234 567.23
Als Fließkommazahl formatiert: 1,234,567.23
Als Fließkommazahl formatiert: 1 234 567.23

```

NumberFormatInfo stellt Ihnen fur die Spezialisierung von Formatierungen im hier gezeigten Stil eine ganze Reihe von Eigenschaften zur Verfugung, die Sie nach Belieben vor dem Einsatz in ToString einstellen konnen. Welche Eigenschaften welche nderung bewirken, entnehmen Sie bitte der Online-Hilfe von Visual Studio.

Gezielte Formatierungen von Zeitwerten mit DateTimeFormatInfo

Was fur die Spezialisierung von numerischen Formatierungen gilt, ist auch fur die Datums- und Zeitenformatierung gultig. Sie verwenden die DateTimeFormatInfo-Klasse, um Formatierungen dieses Datentyps zu spezialisieren.

Auch hier soll ein Beispiel dem besseren Verstandnis dienen. Normalerweise gibt es keine AM/PM-Designatoren (Tagesbereichsbezeichner beim englisch/amerikanischem Datumsformat) fur die deutschen Kultureinstellungen. Das folgende kleine Beispielprogramm richtet die Designatoren mit einem DateTimeFormatInfo-Objekt gezielt mit deutschen Bezeichnungen ein, und sie werden anschlieend durch die Angabe der entsprechenden Formatzeichen in der Formatzeichenkette mit ToString bei der Umwandlung des Datums ausgegeben.

```

Sub main()

    'Zu verwendenden Wert definieren.
    Dim locDate As Date = #12/24/2005 1:12:23 PM#
    'Kultureinstellungen sind deutsch.
    Dim locCultureInfo As New CultureInfo("de-DE")
    'Die Einstellungen, die CultureInfo auf Grund der Kultur schon
    'geleistet hat, ubernehmen wir in ein DateTimeFormatInfo..
    Dim locDateTimeFormatInfo As DateTimeFormatInfo = locCultureInfo.DateTimeFormat

    '...dessen anzuwendende Regeln wir nun genauer spezifizieren konnen:
    locDateTimeFormatInfo.AMDesignator = "Vormittag"
    locDateTimeFormatInfo.PMDesignator = "Nachmittag"
    Console.WriteLine("Mit deutschen AM/PM-Designatoren: " _
        + locDate.ToString("dd.MM.yyyy hh:mm:ss - tt", locDateTimeFormatInfo))
    '12 Stunden dazu addieren:
    locDate = locDate.AddHours(12)
    Console.WriteLine("Mit deutschen AM/PM-Designatoren: " _
        + locDate.ToString("dd.MM.yyyy hh:mm:ss - tt", locDateTimeFormatInfo))
End Sub

```

Kombinierte Formatierungen

Kombinierte Formatierungen sind ein spezielles Feature von .NET, das die Einbindung von zu formatierendem bzw. umzuwandelndem Text gezielt an bestimmte Stellen innerhalb einer Zeichenkette ermöglicht. Dazu ein Beispiel:

Das folgende kleine Programm definiert einen bestimmten Ausgangszeitpunkt. Es addiert anschließend 15 Mal in Folge eine zufällige Zeitspanne zwischen 0 Minuten und 23 Stunden und 59 Minuten zum Ausgangsdatum und zeigt das entsprechende Ergebnis an. Das Programm dafür sieht folgendermaßen aus:

BEGLEITDATEIEN: Im Projekt im Verzeichnis `\VB 2005 - Entwicklerbuch\E - Datentypen\Kap17\CompositeFormatting\` finden Sie die anschließenden Beispiele.

```
Sub Main()
    Dim locBasisDatum As Date = #12/30/2005 1:12:32 PM#
    Dim locOffset As TimeSpan
    Dim locRandom As New Random(Now.Millisecond)
    'Backslash vor ', damit es gedruckt und nicht als Steuerzeichen interpretiert wird!
    Console.WriteLine("Es ist " + _
        locBasisDatum.ToString("dddd, ""der"" d. MMMM \\'yy, HH:mm") + _
        "...")

    '15 Wiederholungen
    For count As Integer = 1 To 15
        locOffset = New TimeSpan(locRandom.Next(23), locRandom.Next(59), 0)
        locBasisDatum = locBasisDatum.Add(locOffset)
        Console.WriteLine("...und " + Math.Floor(locOffset.TotalHours).ToString() + _
            " Std. und " + locOffset.Minutes.ToString() + _
            " Min. später ist " + _
            locBasisDatum.ToString("dddd, ""der"" d. MMMM \\'yy, HH:mm"))
    Next
End Sub
```

Es produziert etwa folgende Ausgabe:

```
Es ist Freitag, der 30. Dezember '05, 13:12...
...und 10 Std. und 21 Min. später ist Freitag, der 30. Dezember '05, 23:33
...und 2 Std. und 29 Min. später ist Samstag, der 31. Dezember '05, 02:02
...und 15 Std. und 16 Min. später ist Samstag, der 31. Dezember '05, 17:18
...und 20 Std. und 10 Min. später ist Sonntag, der 1. Januar '06, 13:28
...und 21 Std. und 43 Min. später ist Montag, der 2. Januar '06, 11:11
...und 2 Std. und 39 Min. später ist Montag, der 2. Januar '06, 13:50
...und 7 Std. und 53 Min. später ist Montag, der 2. Januar '06, 21:43
...und 10 Std. und 34 Min. später ist Dienstag, der 3. Januar '06, 08:17
...und 18 Std. und 52 Min. später ist Mittwoch, der 4. Januar '06, 03:09
...und 13 Std. und 27 Min. später ist Mittwoch, der 4. Januar '06, 16:36
...und 1 Std. und 24 Min. später ist Mittwoch, der 4. Januar '06, 18:00
...und 11 Std. und 40 Min. später ist Donnerstag, der 5. Januar '06, 05:40
...und 20 Std. und 46 Min. später ist Freitag, der 6. Januar '06, 02:26
...und 13 Std. und 31 Min. später ist Freitag, der 6. Januar '06, 15:57
...und 13 Std. und 50 Min. später ist Samstag, der 7. Januar '06, 05:47
```

Zwei Dinge fallen auf: Zum einen ist der Programmtext ein einziges Chaos. Man muss sich Buchstabe für Buchstabe durch das Listing hangeln, um erst nach geraumer Zeit festzustellen, was die Zeilen überhaupt bewirken. Das zweite Problem: Auch die Ausgabe ist nicht wirklich sauber formatiert.

Schauen Sie sich jetzt die folgende Version des Programms an, das die exakt gleiche Ausgabe produziert:

```
Sub ZweiteVersion()

    Dim locBasisDatum As Date = #12/30/2005 1:12:32 PM#
    Dim locOffset As TimeSpan
    Dim locRandom As New Random(Now.Millisecond)

    With locBasisDatum
        Console.WriteLine("Es ist {0}, der {1}...", _
            .ToString("ddd"), _
            .ToString("d. MMMM \'yy, HH:mm"))

        '15 Wiederholungen
        For count As Integer = 1 To 15
            locOffset = New TimeSpan(locRandom.Next(23), locRandom.Next(59), 0)
            locBasisDatum = locBasisDatum.Add(locOffset)
            Console.WriteLine("...und {0} Std. und {1} Min. später ist {2}", _
                Math.Floor(locOffset.TotalHours).ToString(), _
                locOffset.Minutes.ToString(), _
                locBasisDatum.ToString("ddd, "der" d. MMMM \'yy, HH:mm"))
        Next
    End With

End Sub
```

Was ist passiert? Sie haben durch Platzhalter in geschweiften Klammern bestimmt, an welchen Positionen innerhalb des angegebenen Textes, der die Platzhalter enthält, die folgenden Parameter eingesetzt werden sollen. Das Ergebnis kann sich sehen lassen: Das Listing ist leicht lesbar, und man versteht fast auf Anhieb, welchem Zweck es dient.

Sie können kombinierte Formatierungen nicht nur in der `Console.WriteLine`-Anweisung einsetzen. Alle Ableitungen der `TextWriter`-Klasse verstehen mit ihrer `WriteLine`-Anweisung ebenfalls kombinierte Formatierungen. Und zu guter Letzt haben Sie mit der statischen `String.Format`-Methode die Möglichkeit, einen neuen String zu produzieren, der durch die Möglichkeiten der kombinierten Formatierung aufbereitet werden kann.

Ausrichtungen in kombinierten Formatierungen

Die Funktionalität von kombinierten Formatierungen geht noch weiter. Sie können in den so genannten Indexkomponenten – so nennen sich die in geschweiften Klammern eingefügten Platzhalter – angeben, wie der Text mit führenden Leerzeichen ausgerichtet werden soll. Dazu geben Sie mit Komma getrennt die Anzahl der Zeichen ein, auf die der Text durch das Einfügen von führenden Leerzeichen verlängert werden soll, sodass die einzufügenden Zeichen auf einer bestimmten Position enden. Angewendet auf das schon vorhandene Beispiel, ergibt sich folgende Version des Programms:

```

Sub DritteVersion()

    Dim locBasisDatum As Date = #12/30/2005 1:12:32 PM#
    Dim locOffset As TimeSpan
    Dim locRandom As New Random(Now.Millisecond)

    With locBasisDatum
        Console.WriteLine("Es ist {0}, der {1}...", _
            .ToString("ddd"), _
            .ToString("d. MMMM \'yy, HH:mm"))
        '15 Wiederholungen
        For count As Integer = 1 To 15
            locOffset = New TimeSpan(locRandom.Next(23), locRandom.Next(59), 0)
            locBasisDatum = locBasisDatum.Add(locOffset)
            Console.WriteLine("...und {0,2} Std. und {1,2} Min. später ist {2,11}, der {3}", _
                Math.Floor(locOffset.TotalHours).ToString(), _
                locOffset.Minutes.ToString(), _
                .ToString("ddd"), _
                .ToString("dd. MMMM \'yy, HH:mm") _
            )
        Next
    End With
End Sub

```

Wenn Sie diese Prozedur laufen lassen, ergibt sich die folgende Ausgabe auf dem Konsolenfenster:

```

Es ist Freitag, der 30. Dezember '05, 13:12...
...und 4 Std. und 14 Min. später ist Freitag, der 30. Dezember '05, 17:26
...und 13 Std. und 43 Min. später ist Samstag, der 31. Dezember '05, 07:09
...und 11 Std. und 11 Min. später ist Samstag, der 31. Dezember '05, 18:20
...und 2 Std. und 26 Min. später ist Samstag, der 31. Dezember '05, 20:46
...und 4 Std. und 49 Min. später ist Sonntag, der 01. Januar '06, 01:35
...und 16 Std. und 4 Min. später ist Sonntag, der 01. Januar '06, 17:39
...und 3 Std. und 1 Min. später ist Sonntag, der 01. Januar '06, 20:40
...und 15 Std. und 28 Min. später ist Montag, der 02. Januar '06, 12:08
...und 9 Std. und 24 Min. später ist Montag, der 02. Januar '06, 21:32
...und 18 Std. und 18 Min. später ist Dienstag, der 03. Januar '06, 15:50
...und 4 Std. und 26 Min. später ist Dienstag, der 03. Januar '06, 20:16
...und 13 Std. und 17 Min. später ist Mittwoch, der 04. Januar '06, 09:33
...und 0 Std. und 35 Min. später ist Mittwoch, der 04. Januar '06, 10:08
...und 22 Std. und 44 Min. später ist Donnerstag, der 05. Januar '06, 08:52
...und 15 Std. und 38 Min. später ist Freitag, der 06. Januar '06, 00:30

```

Zugegeben: Schöner ist nur die Formatierung der ersten beiden Parameter geworden. Ob das Einrücken der Wochentage wirklich zur Lesbarkeit beiträgt, ist strittig. Zur Demonstration der Funktion ist es allemal ein brauchbares Ergebnis, denn: Sie können leicht erkennen, wie sich die Erweiterung der Indexkomponenten um die Angabe der Gesamtlänge der Buchstaben (die entsprechende Zeile im Listing ist fett markiert) auf das Ergebnis auswirken.

Angeben von Formatzeichenfolgen in den Indexkomponenten

Zu guter Letzt gibt es eine weitere Möglichkeit, die Indexkomponenten zu parametrisieren. Sie können die Formatzeichenfolge, die sich in den bisherigen Versionen bei den zu formatierenden Parametern selbst befand, ebenfalls in jeder Indexkomponente angeben. Dazu trennen Sie die Formatzeichen per Doppelpunkt von den weiteren Parametern. Das folgende Listing zeigt die letzte Version der Prozedur, bei der sie von dieser Möglichkeit Gebrauch macht:

```
Sub VierteVersion()

    Dim locBasisDatum As Date = #12/30/2005 1:12:32 PM#
    Dim locOffset As TimeSpan
    Dim locRandom As New Random(Now.Millisecond)

    Console.WriteLine("Es ist {0:ddd}, der {1:d. MMMM \'yy, HH:mm}...", _
        locBasisDatum, _
        locBasisDatum)

    '15 Wiederholungen
    For count As Integer = 1 To 15
        locOffset = New TimeSpan(locRandom.Next(23), locRandom.Next(59), 0)
        locBasisDatum = locBasisDatum.Add(locOffset)
        Console.WriteLine("...und {0,2} Std. und {1,2} Min. später ist {2,11:ddd}, der {3:dd. MMM \'yy, HH:mm}", _
            Math.Floor(locOffset.TotalHours).ToString(), _
            locOffset.Minutes.ToString(), _
            locBasisDatum, _
            locBasisDatum
        )
    Next
End Sub
```

WICHTIG: Achten Sie bei dieser Version auf zwei wesentliche Änderungen. Da zum einen die Formatierungsanweisungen in die Indexkomponenten verschoben wurden, dürfen Sie jetzt nicht mehr die ToString-Funktion der einzelnen zu formatierenden Daten verwenden, da diese zuvor für die korrekte Formatierung zuständig war. Würden Sie die ToString-Funktion beibehalten, so würde die Formatierungsfunktion, die durch WriteLine ins Leben gerufen wird (der so genannte *Formatter*), versuchen, die Formatzeichen auf eine Zeichenkette und nicht auf den Date-Typ anzuwenden. Er würde natürlich ins Leere laufen, da Zeichenketten selbst nicht mit den Formatzeichen für den Date-Typ zu formatieren sind.

Beachten Sie auch, dass das bündige Ausrichten durch Leerzeichen nur mit nicht-proportionalen Zeichensätzen möglich ist. Bei proportionalen Zeichensätzen, bei denen Buchstaben nicht gleich groß sind («www» ist viel länger als «iii», mit anderen Worten: Ich schreibe gerade in einer proportionalen Schrift) schlägt das Formatieren mit Leerzeichen natürlich fehl.

So helfen Ihnen benutzerdefinierte Format Provider, Ihre Programme zu internationalisieren

.NET erlaubt das Erstellen von benutzerdefinierten Format Providern. Um zu verstehen, wie Sie Format Provider in .NET entwickeln, lassen Sie mich das Pferd anhand des Ergebnisses eines Beispiels von hinten aufzäumen: Stellen Sie sich vor, Sie müssten ein Programm entwickeln, das nicht nur die Konvertierung von Maßeinheiten vornehmen kann, sondern den Entwickler seiner Klasse auch bei der Aufbereitung der Werte unterstützt.

Ein Programm soll folgendes Problem lösen: Es erlaubt seinem Anwender, einen Wert in Meter einzugeben; das Programm wird anschließend die Werte in die in deutsch- und englischsprachigen Kulturen üblichen Maßeinheiten umrechnen, etwa wie im folgenden Beispiel zu sehen:

```
Geben Sie einen Wert in Metern zur Umrechnung ein: 550
      mm      |      cm      |      m      |      km      |
550.000,00000 | 55.000,00000 | 550,00000  | 0,55000    |

      lines   |      inches  |      yards  |      miles   |
259.820,00000 | 21.653,50000 | 601,70000  | 0,34177    |
```

Soweit ist das Programm absolut nichts Besonderes – jeder Basic-Neueinsteiger programmierte so etwas schon vor Jahrzehnten nach ein paar Stunden.

Allerdings ist der Weg dorthin schon bemerkenswerter. Das folgende Programm zeigt, wie die einzelnen Zeilen zustande gekommen sind.

BEGLEITDATEIEN: Sie finden dieses Programm im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap17\CustomFormatProvider01\`.

```
Module CustomFormatProvider
```

```
    Sub Main()
```

```
        Dim locEngKultur As New CultureInfo("en-US")
```

```
        Console.WriteLine("Geben Sie einen Wert in Metern zur Umrechnung ein: ")
```

```
        Dim locLaenge As New Laengen(Decimal.Parse(Console.ReadLine))
```

```
        'Umgerechneten Wert anzeigen:
```

```
        Console.WriteLine("      mm      |      cm      |      m      |      km      |")
```

```
        Console.WriteLine("{0,17} |{1,17} |{2,17} |{3,17} |", _
```

```
            locLaenge.ToString("s-d;#,##0.00000"), _
```

```
            locLaenge.ToString("k-d;#,##0.00000"), _
```

```
            locLaenge.ToString("m-d;#,##0.00000"), _
```

```
            locLaenge.ToString("g-d;#,##0.00000"))
```

```
        Console.WriteLine()
```

```
        'Umgerechneten Wert anzeigen:
```

```
        Console.WriteLine("      lines   |      inches  |      yards  |      miles   |")
```

```
        Console.WriteLine("{0,17} |{1,17} |{2,17} |{3,17} |", _
```

```
            locLaenge.ToString("s;#,##0.00000", locEngKultur), _
```

```

    locLaenge.ToString("k-e;#,##0.00000"), _
    locLaenge.ToString("m;#,##0.00000", _
        New LaengenFormatInfo(LaengenKultur.EnglischAmerikanisch, LaengenAufloesung.Mittel)), _
    locLaenge.ToString("g;#,##0.00000", locEngKultur))
Console.ReadLine()

```

End Sub

End Module

Sie können leicht erkennen, dass Sie in diesem Programm quasi keine einzige Berechnung finden können. Es werden auch keine Eigenschaften oder Funktionen einer speziellen Klasse aufgerufen – alle Konvertierungen finden ausschließlich über die Steuerung entweder von Formatzeichenfolgen oder – was auf den ersten Blick noch nicht offensichtlich ist – über mehr oder weniger spezielle Format Provider statt.

Die Konvertierungen werden aber nichtsdestotrotz durch eine spezielle Klasse realisiert – sie nennt sich für dieses Beispiel schlicht und einfach *Laengen*. Dies ist eine von mir selbst geschriebene Klasse, suchen Sie sie also nicht im .NET Framework, dazu später mehr. Sie stellt auf der einen Seite die wirklich simplen Konvertierungsfunktionen zur Verfügung (mit Methoden wie *ToMile*, *ToInch*, etc.). Auf der anderen Seite erweitert sie die *ToString*-Funktion der Basisklasse (sie ist direkt von *Object* abgeleitet). *ToString* nimmt wahlweise einen oder zwei Parameter entgegen, und zwar in dem Stil, den Sie in den vergangenen Abschnitten schon kennen gelernt haben. Sie verarbeitet Formatzeichenfolgen – wahlweise in Kombination mit einem Format Provider.

Die Formatzeichenfolgen akzeptieren Formatzeichenfolgen, die eigentlich Kombinationen aus zweien sind – oder zumindest sein können. Der erste Teil einer Formatzeichenfolge steuert, welche Maßeinheit bei der Ausgabe verwendet werden soll; der zweite Teil – und jetzt kommt der internationalisierende Part – bestimmt, welche Kulturvorgaben dabei verwendet werden sollen. Aus diesem Grund bestimmen Sie mit der Angabe von den Formatzeichenfolgen auch nicht direkt »Zentimeter« oder »Inch« (die englische Einheit für Zoll), sondern geben vielmehr eine Skalierungsbezeichnung an, wahlweise in Kombination mit einem Kulturbuchstaben. Die Skalierung habe ich der Einfachheit halber in *sehr klein*, *klein*, *mittel* und *groß* festgelegt. Diese Version der Klasse unterscheidet ferner die beiden Kulturen *deutsch* und *amerikanisch-englisch*.

Damit haben Sie, ohne direkte Bezeichnungen definieren zu müssen, kulturabhängig die Möglichkeit, verschiedene Skalierungen zu verwenden. Mit den folgenden Anweisungen lassen Sie beispielsweise eine Zeile ausgeben, die die *sehr kleine* Skalierung mit einem Wert für die Klasse *Laenge* von 1 verwendet:³

Sub Spielchen()

```

    'Definiert eine Laengen-Instanz mit 1 (einem Meter).
    Dim locLaenge As New Laengen(1)
    'Gibt auf einem deutschen System den Klasseninstanzwert in Millimeter aus.
    Console.WriteLine(locLaenge.ToString("s"))

```

³ Die einzelnen Testroutinen befinden sich alle innerhalb des Moduls. Wenn Sie sie selber ausprobieren wollen, brauchen Sie am Anfang des Moduls lediglich die Kommentare der ersten beiden Zeilen zu entfernen und den Aufruf der jeweils vorgestellten Prozedur dort einzusetzen.

```

'Auf einem englischen oder amerikanischen System würde die vorherige Zeile
'die gleiche Ausgabe, wie die folgende bewirken:
Console.WriteLine(locLaenge.ToString("s", New CultureInfo("en-US")))
Console.ReadLine()
End Sub

```

Dieses Programm gibt die folgenden Zeilen aus:

```

1000
472,4

```

Sehr klein im Deutschen bedeutet bei diesem Beispiel *Millimeter*. Da die Klasse in der Einheit *Meter* definiert wird, druckt die entsprechende Programmzeile korrekt 1000 (für 1000 Millimeter) aus und in der englischen Version, in der *sehr klein* die Einheit *Lines* bedeutet, korrekt 472,4.

Welche Kultur Sie bei der Ausgabe berücksichtigen, lässt sich bei der *Laengen*-Klasse nicht nur mit dem *CultureInfo*-Objekt steuern, wie im letzten Beispiel gesehen. Sie haben nämlich auch die Möglichkeit, die Kultur in einer Erweiterung der Formatzeichenfolge zu bestimmen, etwa wie im folgenden Beispiel, das kein *CultureInfo*-Objekt verwendet:

```

Sub Spielchen2()

'Definiert eine Laengen-Instanz mit 1 (einem Meter).
Dim locLaenge As New Laengen(1)
'Gibt auf jedem System den Klasseninstanzwert in Millimeter aus.
Console.WriteLine(locLaenge.ToString("s-d"))
'Gibt auf jedem System den Klasseninstanzwert in Lines aus.
Console.WriteLine(locLaenge.ToString("s-e"))

Console.ReadLine()
End Sub

```

Die Ausgabe ist dieselbe wie im vorherigen Beispiel.

Die Steuerung mit Formatzeichen erlaubt aber noch mehr. Mit Semikolon getrennt können Sie eine Formatierung für die Werte an sich bestimmen, wie Sie es im ► Abschnitt »Formatierung von numerischen Ausdrücken durch Formatzeichenfolgen« ab Seite 501 schon kennen gelernt haben. Ein weiteres Beispiel zeigt diese Verwendung der Formatzeichen:

```

Sub Spielchen3()

'Definiert eine Laengen-Instanz mit 1 (einem Meter).
Dim locLaenge As New Laengen(1)
'Gibt auf jedem System den Klasseninstanzwert in Millimeter aus.
Console.WriteLine(locLaenge.ToString("s-d;#,##0.00"))
'Gibt auf jedem System den Klasseninstanzwert in Lines aus.
Console.WriteLine(locLaenge.ToString("s-e;#,##0.00"))

Console.ReadLine()
End Sub

```

Die Ausgabe lautet jetzt:

```

1.000,00
472,40

```

die Zahlen wurden den Formatzeichen entsprechend formatiert.

Die folgende Tabelle zeigt Ihnen, welche Zeichenkombinationen die ToString-Funktion meiner Laengen-Klasse auswerten kann:

Formatzeichen	Bedeutung	deutsche Maßeinheit (-d)	englisch-amerikanische Maßeinheit (-e)
s	sehr klein	Millimeter	Lines
k	klein	Zentimeter	Inches
m	mittel	Meter	Yard
g	groß	Kilometer	Miles

Tabelle 17.6: Die *Laengen*-Klasse versteht diese Formatzeichen

Bislang haben Sie von eigentlichen benutzerdefinierten Format Providern noch nichts gesehen. Die Formatsteuerung mit Formatzeichen in Kombination mit dem CultureInfo-Objekt schien zwar schon ganz gut zu funktionieren, aber die Ausgabeform eines Laengen-Klassenwertes konnten Sie nur auf Kulturseite beeinflussen.

Zusätzlich zur Laengen-Klasse gibt es im Beispielprogramm aber auch einen richtigen Format Provider – er nennt sich passenderweise LaengenFormatInfo. Seine Funktionsweise demonstriert das folgende Beispiel:

```
Sub Spielchen4()

    'Definiert eine Laengen-Instanz mit 1 (einem Meter).
    Dim locLaenge As New Laengen(1)
    Dim locLaengenFormatInfo As New LaengenFormatInfo

    locLaengenFormatInfo.Aufloesung = LaengenAufloesung.SehrKlein
    locLaengenFormatInfo.Kultur = LaengenKultur.Deutsch

    'Gibt auf jedem System den Klasseninstanzwert in Millimeter aus.
    Console.WriteLine(locLaenge.ToString(locLaengenFormatInfo))
    'Gibt auf jedem System den Klasseninstanzwert in Lines aus.
    locLaengenFormatInfo.Kultur = LaengenKultur.EnglischAmerikanisch
    Console.WriteLine(locLaenge.ToString(locLaengenFormatInfo))

    Console.ReadLine()

End Sub
```

Die Verwendungsweise lehnt sich an die bekannten Format Provider NumberFormatInfo und DateTimeFormatInfo an. Sie instanzieren die Klasse, setzen bestimmte Eigenschaften (im Beispiellisting fett gekennzeichnet), und wenn Sie die Klasseninstanz der ToString-Funktion der Laengen-Klasse übergeben, passt sie die Formatierung des Ausgabertextes entsprechend an.

Das Ergebnis dieses Beispiels ist wieder das des vorherigen.

Nachdem Sie die Laengen-Klasse nun umfassend anzuwenden gelernt haben, will ich Ihnen die genaue Funktionsweise nicht vorenthalten.

Die Klasse besteht zunächst einmal aus dem Konstruktor, und einer ganzen Menge einfacher Umrechnungsfunktionen, die folgendermaßen implementiert sind:

Public Class Laengen

```
'Speichert die Länge in Meter.  
Private myLaenge As Decimal
```

```
Sub New(ByVal Meter As Decimal)  
    myLaenge = Meter  
End Sub
```

```
Public Shared Function FromMile(ByVal Mile As Decimal) As Laengen  
    Return New Laengen(Mile * 1609D)  
End Function
```

```
Public Shared Function FromYard(ByVal Yard As Decimal) As Laengen  
    Return New Laengen(Yard * 0.9144D)  
End Function
```

```
Public Shared Function FromInch(ByVal Inch As Decimal) As Laengen  
    Return New Laengen(Inch * 0.0254D)  
End Function
```

```
Public Shared Function FromLine(ByVal Line As Decimal) As Laengen  
    Return New Laengen(Line * 0.002117D)  
End Function
```

```
Public Shared Function FromKilometer(ByVal Kilometer As Decimal) As Laengen  
    Return New Laengen(Kilometer * 1000D)  
End Function
```

```
Public Shared Function FromCentimeter(ByVal Centimeter As Decimal) As Laengen  
    Return New Laengen(Centimeter * 0.01D)  
End Function
```

```
Public Shared Function FromMillimeter(ByVal Millimeter As Decimal) As Laengen  
    Return New Laengen(Millimeter * 0.001D)  
End Function
```

```
Public Function ToMeter() As Decimal  
    Return myLaenge  
End Function
```

```
Public Function ToKilometer() As Decimal  
    Return myLaenge * 0.001D  
End Function
```

```
Public Function ToCentimeter() As Decimal  
    Return myLaenge * 100D  
End Function
```

```
Public Function ToMillimeter() As Decimal  
    Return myLaenge * 1000D  
End Function
```

```

Public Function ToMile() As Decimal
    Return myLaenge * 0.0006214D
End Function

Public Function ToYard() As Decimal
    Return myLaenge * 1.094D
End Function

Public Function ToInch() As Decimal
    Return myLaenge * 39.37D
End Function

Public Function ToLine() As Decimal
    Return myLaenge * 472.4D
End Function

```

Das Interessante sind anschließend die verschiedenen Überladungen der ToString-Funktionen, mit denen der Inhalt der Klasseninstanz in Zeichenketten umgewandelt werden kann:

```

Public Overloads Function ToString(ByVal format As String) As String
    Return ToString(format, Nothing)
End Function

Public Overloads Function ToString(ByVal formatProvider As System.IFormatProvider) As String
    Return ToString(Nothing, formatProvider)
End Function

Public Overloads Function ToString(ByVal formatChars As String, _
    ByVal formatProvider As System.IFormatProvider) As String

    Trace.WriteLine("ToString (Formattable-Signatur) wurde aufgerufen!")

    If (TypeOf formatProvider Is CultureInfo) Or formatProvider Is Nothing Then
        formatProvider = LaengenFormatInfo.FromFormatProvider(formatProvider)
    ElseIf Not (TypeOf formatProvider Is LaengenFormatInfo) Then
        Dim up As New _
            FormatException("Der Format Provider wird für die Klasse Laengen nicht unterstützt!")
        Throw up
    End If

    'LaengenFormatInfo-Provider enthält die Format-Aufbereitungsroutine
    Return DirectCast(formatProvider, LaengenFormatInfo).Format(formatChars, Me, formatProvider)

End Function

End Class

```

Erwähnenswert an dieser Stelle ist die Vorgehensweise zum Erkennen des Typs des übergebenen Format Providers am Anfang des Listings. Hier wird nämlich kein fester Typ als Parameter übernommen, sondern eine Schnittstelle. Eine Schnittstelle deswegen, damit wahlweise ein CultureInfo-Objekt, ein LaengenFormatInfo-Objekt oder Nothing übergeben werden kann. Mit Type Of überprüft ToString dabei, um welchen Typ es sich bei der Schnittstelle genau handelt (die relevanten Stellen sind im Listing wieder fett markiert). Sollte Nothing oder eine CultureInfo übergeben worden sein,

dann kümmert sich die statische Methode `FromFormatProvider` der `LaengenFormatInfo`-Klasse darum, dass im Anschluss nur mit eben diesem Format Provider gearbeitet wird. Und eigentlich macht genau diese Funktionsweise die Internationalisierung des Programms aus: Wenn kein Format Provider übergeben wurde, dann legt – wie wir später noch sehen werden – die statische Funktion nicht etwa direkt ein `LaengenFormatInfo`-Objekt an, sondern zunächst ein `CultureInfo`-Objekt. Dieses `CultureInfo`-Objekt ist aber nicht irgendeins, sondern es spiegelt die voreingestellte Kultur des aktuellen Threads wider – und damit die Grundeinstellung des Rechners. Erst jetzt passiert die Konvertierung in ein `LaengenFormatInfo`-Objekt – mit dem Ergebnis, dass auf einem englischen System automatisch englische und auf einem deutschen System automatisch deutsche Maßeinheiten verwendet werden.

Die Formatierungsroutine selbst befindet sich ebenfalls in unserem `LaengenFormatInfo`-Objekt. Da das zu diesem Zeitpunkt, zu dem wir es zum Aufruf von `Format` benötigen, aber unbedingt vorhanden ist (alle anderen möglicherweise artfremden Format Provider sind zu diesem Zeitpunkt entweder konvertiert worden oder haben eine Ausnahme ausgelöst), können wir die Schnittstellenvariable `formatProvider`, ohne eine Ausnahme zu riskieren, in ein `LaengenFormatInfo`-Objekt casten, um uns den Zugang zu dessen `Format`-Methode zu erschließen.

Die komplette Aufbereitungsfunktionalität an sich findet anschließend in der `LaengenFormatInfo`-Klasse in eben dieser Funktion statt. Diese Klasse ist im folgenden Listing zu sehen.

```
Public Enum LaengenAufloesung
    SehrKlein ' Millimeter oder Line
    Klein     ' Zentimeter oder Inch
    Mittel    ' Meter oder Yard
    Groß     ' Kilometer oder Mile
End Enum
```

```
Public Enum LaengenKultur
    EnglischAmerikanisch
    Deutsch
End Enum
```

Diese beiden Enums (mehr zu Enums im nächsten Kapitel) dienen lediglich dazu, dem Entwickler den Umgang mit der im Anschluss besprochenen `LaengenFormatInfo`-Klasse zu erleichtern; er muss sich dann keine Nummern für Parametereinstellungen merken, sondern kann per Namen darauf zugreifen. Die eigentliche `LaengenFormatInfo`-Klasse verwendet die Enums an verschiedenen Stellen.

```
Public Class LaengenFormatInfo
    Implements IFormatProvider
    Private myKultur As LaengenKultur
    Private myAufloesung As LaengenAufloesung

    Sub New()
        myKultur = LaengenKultur.Deutsch
        myAufloesung = LaengenAufloesung.Mittel
    End Sub

    Sub New(ByVal Kultur As LaengenKultur)
        myKultur = Kultur
        myAufloesung = LaengenAufloesung.Mittel
    End Sub
```

```

Sub New(ByVal Kultur As LaengenKultur, ByVal Aufloesung As LaengenAufloesung)
    myKultur = Kultur
    myAufloesung = Aufloesung
End Sub
Public Shared Function FromFormatProvider(ByVal formatProvider As IFormatProvider) As LaengenFormatInfo

    Dim retLaengenFormatInfo As LaengenFormatInfo

    If formatProvider Is Nothing Then
        formatProvider = CultureInfo.CurrentCulture
    End If

    If DirectCast(formatProvider, CultureInfo).ThreeLetterISOLanguageName = "deu" Then
        retLaengenFormatInfo = _
            New LaengenFormatInfo(LaengenKultur.Deutsch, LaengenAufloesung.Mittel)
    Else
        retLaengenFormatInfo = _
            New LaengenFormatInfo(LaengenKultur.EnglischAmerikanisch, LaengenAufloesung.Mittel)
    End If
    Return retLaengenFormatInfo

End Function

```

Erste Station: die schon angesprochene statische Funktion `FromFormatProvider`, die dafür sorgt, dass jeder ankommende Format Provider automatisch in einen `LaengenFormatInfo`-Format-Provider umgewandelt wird. Sie sorgt dafür – wie schon gesagt –, dass die Klasse `Laengen` sich ohne weiteres Eingreifen durch den Entwickler automatisch internationalisiert.

```

Public Function GetFormat(ByVal formatType As System.Type) As Object _
    Implements System.IFormatProvider.GetFormat

    Trace.WriteLine("Ausgabe von GetFormat:" + formatType.Name)

End Function

```

Da durch die `Implements`-Anweisung am Anfang der Klasse die `IFormatProvider`-Schnittstelle eingebunden wird, muss diese Funktion `GetFormat` ebenfalls vorhanden sein. Momentan enthält sie nur eine einzelne Anweisung, die – sollte sie von wem oder was auch immer aufgerufen werden – uns darüber im Ausgabefenster (nicht Konsolenfenster!) informieren wird. Dadurch, dass wir die `IFormatProvider`-Schnittstelle einbinden, ermöglichen wir, sie auch als Parameter für `ToString` der `Laengen`-Klasse zuzulassen. Auf diese Weise schaffen wir die Möglichkeit, die Schnittstelle zu übergeben, ohne uns bei der Parameterübergabe ausschließlich auf ein `LaengenInfoFormat`-Objekt festlegen zu müssen.

TIPP: Wenn Sie das Kapitel über Schnittstellen (noch) nicht gelesen haben, dann beachten Sie Folgendes, um die benötigten Routinenrumpfe (Stubs) für die Schnittstelle `IFormatProvider` einzufügen, falls Sie ähnlichen Code selber erstellen: Tippen Sie **Implements IFormatProvider** und gehen Sie in die nächste Zeile mit **Eingabe** (nicht mit den Cursortasten!). Fertig.

```

Public Function Format(ByVal formatChars As String, _
    ByVal arg As Object, _
    ByVal formatProvider As System.IFormatProvider) As String _

```

```

Dim locLaengen As Laengen

Trace.WriteLine("Format (CustomFormatter-Signatur) wurde aufgerufen!")
'Dafür sorgen, dass das zu formatierende Element und der Format Provider übereinstimmen.
If Not TypeOf arg Is Laengen Then
    Return String.Format(formatProvider, formatChars, arg)
End If

locLaengen = DirectCast(arg, Laengen)

'Dafür sorgen, dass die Formatzeichenfolge nie "nichts" ist.
If formatChars Is Nothing Then
    formatChars = ""
End If

'Mit Semikolon können Formatzeichen für die Formatierung des eigentlichen Wertes folgen.
Dim locSemikolonPos As Integer = formatChars.IndexOf(";")

'Standardzeichen für die Formatzeichen zur Werteformatierung vorgeben.
Dim locNumFormat As String = "G"

'Doppelpunkt gefunden.
If locSemikolonPos > -1 Then
    'Das ist die Formatzeichenfolge für die Werteformatierung
    locNumFormat = formatChars.Substring(locSemikolonPos + 1)

    'Das für die Wahl der Längeneinheit
    formatChars = formatChars.Substring(0, locSemikolonPos)

    'Leerstring kommt nicht in Frage.
    If locNumFormat = "" Then
        locNumFormat = "G"
    End If
End If

'Nur noch kein, ein oder drei Zeichen kommen in Frage.
If formatChars.Length <> 0 And formatChars.Length <> 1 And formatChars.Length <> 3 Then
    Dim up As New FormatException("Ungültige(s) Formatzeichen für die Kulturbestimmung!")
    Throw up
End If

'Wenn drei Zeichen, dann wird die Einstellung des FormatProviders ignoriert;
'das Formatzeichen ist der Bestimmer!
If formatChars.Length = 3 Then
    If formatChars.ToUpper.EndsWith("-D") Then
        formatProvider = New LaengenFormatInfo(LaengenKultur.Deutsch)
        formatChars = formatChars.Substring(0, 1)
    ElseIf formatChars.ToUpper.EndsWith("-E") Then
        formatProvider = New LaengenFormatInfo(LaengenKultur.EnglischAmerikanisch)
        formatChars = formatChars.Substring(0, 1)
    Else
        Dim up As New FormatException("Ungültiges Formatzeichen für die Kulturbestimmung!")
        Throw up
    End If
End If

```

```

    End If
End If

'Zu diesem Zeitpunkt ist formatProvider unbedingt eine LaengenformatInfo,
'Das folgende Casting kann also nicht schiefgehen:
Dim locLaengenFormatInfo As LaengenFormatInfo = DirectCast(formatProvider, LaengenFormatInfo)

'formatChars besteht aus (jetzt noch) nur einem Zeichen.

If formatChars.Length = 1 Then
    'S' für 'Sehr klein'
    If formatChars.ToUpper.StartsWith("S") Then
        locLaengenFormatInfo.Aufloesung = LaengenAufloesung.SehrKlein
    End If

    'K' für 'klein'
    If formatChars.ToUpper.StartsWith("K") Then
        locLaengenFormatInfo.Aufloesung = LaengenAufloesung.Klein
    End If

    'M' für 'Mittel'
    If formatChars.ToUpper.StartsWith("M") Then
        locLaengenFormatInfo.Aufloesung = LaengenAufloesung.Mittel
    End If

    'G' für 'groß'
    If formatChars.ToUpper.StartsWith("G") Then
        locLaengenFormatInfo.Aufloesung = LaengenAufloesung.Groß
    End If
End If

With locLaengenFormatInfo
    'Und alle Stringausgaben anhand des Providers durchführen
    If .Kultur = LaengenKultur.Deutsch Then
        If .Aufloesung = LaengenAufloesung.SehrKlein Then
            Return locLaengen.ToMillimeter.ToString(locNumFormat)
        ElseIf .Aufloesung = LaengenAufloesung.Klein Then
            Return locLaengen.ToCentimeter.ToString(locNumFormat)
        ElseIf .Aufloesung = LaengenAufloesung.Mittel Then
            Return locLaengen.ToMeter.ToString(locNumFormat)
        ElseIf .Aufloesung = LaengenAufloesung.Groß Then
            Return locLaengen.ToKilometer.ToString(locNumFormat)
        End If
    Else
        If .Aufloesung = LaengenAufloesung.SehrKlein Then
            Return locLaengen.ToLine.ToString(locNumFormat)
        ElseIf .Aufloesung = LaengenAufloesung.Klein Then
            Return locLaengen.ToInch.ToString(locNumFormat)
        ElseIf .Aufloesung = LaengenAufloesung.Mittel Then
            Return locLaengen.ToYard.ToString(locNumFormat)
        ElseIf .Aufloesung = LaengenAufloesung.Groß Then
            Return locLaengen.ToMile.ToString(locNumFormat)
        End If
    End If
End With

```

```

    End If
  End With
End Function

```

Und hier findet sie nun statt, die Aufbereitung der Zeichenkette für die formatierte Ausgabe. Sie macht unseren Format Provider erst wirklich zu einem *Format* Provider. Doch im Grunde genommen sind die knapp 100 Zeilen, in denen die Aufbereitung des Wertes und die Auswertung der Formatzeichenfolgen stattfindet, nichts Besonderes. Die eine oder andere Zeichenkettenanalyse, ein paar Bedingungsauswertungen – das war es schon.

```

Public Property Kultur() As LaengenKultur
  Get
    Return myKultur
  End Get
  Set(ByVal Value As LaengenKultur)
    myKultur = Value
  End Set
End Property
Public Property Aufloesung() As LaengenAufloesung
  Get
    Return myAufloesung
  End Get
  Set(ByVal Value As LaengenAufloesung)
    myAufloesung = Value
  End Set
End Property
End Class

```

Und damit ist das Geheimnis der Funktionsweise unserer Laengen-Klasse und ihres eigenen Format Providers auch schon gelüftet. Eine Sache fehlt allerdings noch, und die hat es in sich:

Benutzerdefinierte Format Provider durch IFormatProvider und ICustomFormatter

Was bislang kein Beispielprogramm gezeigt hat, ist ein Feature, auf das Sie bei der Formatierung von Datumswerten und numerischen Daten zurückgreifen können: die direkte Einbindung von Formatzeichenfolgen beispielsweise in `WriteLine` oder `String.Format`. Da wir im jetzigen Stand einen Format Provider implementiert haben, schauen wir, was passiert, wenn wir diesen Format Provider zusammen mit unserem Datentyp in einer solchen Kombination einsetzen:

BEGLEITDATEIEN: Sie finden dieses Projekt im Verzeichnis `.\VB 2005 - Entwicklerbuch\E - Datentypen\Kap17\Custom-FormatProvider02`.

```

Sub FormatterTest()

  'Definiert eine Laengen-Instanz mit 1 (einem Meter).
  Dim locLaenge As New Laengen(1)
  Dim locLaengenFormatInfo As New LaengenFormatInfo

  locLaengenFormatInfo.Aufloesung = LaengenAufloesung.SehrKlein
  locLaengenFormatInfo.Kultur = LaengenKultur.EnglischAmerikanisch

```

```

'Testen der Format-Funktion.
Dim locStr As String = String.Format(locLaengenFormatInfo, _
    "Testausgabe {0:; #.##0.00} eines Laengen-Objektes", _
    locLaenge)
Console.WriteLine(locStr)

'Testen des Formatters.
Console.WriteLine("Testausgabe {0:g-d; #.##0.00} eines Laengen-Objektes", locLaenge)

Console.ReadLine()

End Sub

```

Doch leider passiert nach dem Programmstart nicht das, was wir eigentlich erwarten. Die Ausgabe lautet schlicht:

```

Testausgabe CustomFormatProvider.Laengen eines Laengen-Objektes
Testausgabe CustomFormatProvider.Laengen eines Laengen-Objektes

```

Anstatt die formatierten Werte auszugeben, haben beide Zeile lediglich den Namen des Objektes in das Konsolenfenster geschrieben. Allerdings: Im Ausgabefenster (nicht im Konsolenfenster!) erscheint ein Hinweis, der uns bei der Lösung dieses Problems einen Schritt weiter bringt.

Dort ist nämlich

```

GetFormat (IFormatProvider-Signatur) wurde aufgerufen:ICustomFormatter
zu lesen, und genau diese Zeile drucken wir durch die Anweisung
Trace.WriteLine("GetFormat (IFormatProvider-Signatur) wurde aufgerufen:" + formatType.Name)

```

in der GetFormat-Methode der LaengenFormatInfo-Klasse aus. Die Frage, die sich jetzt stellt: *Was hat diese Funktion aufgerufen und warum?*

Um genau zu sein: Die AppendFormat-Methode des StringBuilder-Objektes, die .NET-intern für die Aufbereitung einer zu formatierenden Zeichenfolge mithilfe eines Format Providers zuständig ist, war für den Aufruf von GetFormat verantwortlich. Sie selbst wurde über den Umweg String.Format aufgerufen und fragt uns, welcher benutzerdefinierte Format Provider die eigentliche Formatierung vornehmen soll. Da unsere Routine GetFormat z.Z. noch Nothing zurückgibt (genau genommen gibt sie gar nichts zurück – aber das entspricht ja buchstäblich *Nothing* ...), interpretiert sie das als Antwort »keiner«. Sie greift deswegen auf ein Notprogramm zurück und ruft die parameterlose ToString-Funktion des aufzubereitenden Objektes (Instanz von Laengen) auf. Wir haben diese aber nicht überschrieben, und deswegen liefert sie – da von Object abgeleitet und somit unverändert übernommen – nur den Namen der Klasse zurück, und genau das haben wir als Ausgabe im Konsolenfenster sehen können.

Nur wenn wir innerhalb von GetFormat einen für unseren Datentyp gültigen Formatter zurückliefern, wird AppendFormat diesen verwenden und anschließend dessen Format-Routine aufrufen. Unsere Aufgabe ist es also lediglich, einen Formatter zu implementieren und eine Instanz als Funktionsergebnis von GetFormat an AppendFormat zurückzuliefern. AppendFormat weiß dann, wen es zur Formatierung heranziehen soll. Die einzige sinnvolle Instanz, die GetFormat zu dieser Zeit kennt, ist aber die eigene.

Das bedeutet, dass die `LaengenFormatInfo`-Klasse auch die `ICustomFormatter`-Schnittstelle implementieren muss, damit sie zum `Formatter` wird und ein erlaubtes Funktionsergebnis zurückliefern kann.

Einige Handgriffe reichen, um zum Ziel zu gelangen: Am Anfang des Codes der `LaengenFormatInfo`-Klasse muss die `Implements`-Anweisung um die `IFormatProvider`-Schnittstelle ergänzt werden:

```
Public Class Laengen
    Implements IFormatProvider

    'Speichert die Länge in Meter.
    Private myLaenge As Decimal
    .
    .
    .
```

Die `IFormatProvider`-Schnittstelle verlangt, dass eine `Format`-Funktion in der Klasse existiert, die die Formatierung durchführt. Sie muss als Signatur einen `String` mit Formatzeichen und das zu formatierende Objekt sowie ein weiteres Objekt entgegennehmen, das die `IFormatProvider`-Schnittstelle implementiert. Zufälligerweise⁴ haben wir genau so eine Version von `Format` schon im Programm. Es reicht, diese Funktion mit der `Implements`-Anweisung zu versehen, damit das Implementieren der `IFormatable`-Schnittstelle abgeschlossen ist:

```
Public Function Format(ByVal formatChars As String, _
    ByVal arg As Object, _
    ByVal formatProvider As System.IFormatProvider) As String _
    Implements ICustomFormatter.Format

    Trace.WriteLine("Format (CustomFormatter-Signatur) wurde aufgerufen!")
    'Dafür sorgen, dass das zu formatierende Element und der FormatProvider übereinstimmen
    If Not TypeOf arg Is Laengen Then
        Return String.Format(formatProvider, formatChars, arg)
    End If
    .
    .
    .
```

Und zu guter Letzt teilen wir der `AppendFormat`-Methode, die `GetFormat` der `LaengenFormatInfo` aufruft, noch mit, dass sie unsere `LaengenFormatInfo`-Klasse als `Formatter` verwenden kann. Das machen wir folgendermaßen:

```
Public Function GetFormat(ByVal formatType As System.Type) As Object _
    Implements System.IFormatProvider.GetFormat
    Trace.WriteLine("Ausgabe von GetFormat:" + formatType.Name)
    'Wird mein Typ verlangt?
    If formatType.Name = "ICustomFormatter" Then
        'Ja, diese Instanz ist erlaubt zu handeln!
        Return Me
    Else
        'Falscher Typ, diese Instanz darf nichts machen, denn
        'wenn sie als Provider einem nicht kompatiblen Typ
        'übergeben wird, geht's in die Hose.
        Return Nothing
    End If
End Function
```

⁴ ;-)- vielleicht nicht ganz so zufällig...

```
End If
End Function
```

Wenn wir das Programm nun abermals starten, sieht das Ergebnis viel besser aus:

```
Testausgabe 472,40000 eines Laengen-Objektes
Testausgabe CustomFormatProvider.Laengen eines Laengen-Objektes
```

String.Format liefert jetzt brauchbare Ergebnisse – WriteLine direkt allerdings noch nicht.

Wir haben im vergangenen Abschnitt gesehen, dass AppendFormat sich richtig Mühe gibt, korrekte Formatierungen auch mit Objekten durchzuführen, die dem Framework fremd sind. Und diese Bemühungen gehen noch einen Schritt weiter. Denn bevor alle Stricke reißen und AppendFormat auf die parameterlose ToString-Funktion eines Objektes zurückgreift, sucht es automatisch nach einer weiteren Schnittstelle, die das Objekt implementieren könnte. Ihr Name: IFormattable

Automatisch formatierbare Objekte durch Einbinden von IFormattable

Die IFormattable-Schnittstelle ist in ihrer Handhabung eigentlich sogar noch einfacher als die Format-Provider-Methode – leider auch nicht ganz so flexibel, denn sie muss auf eine korrelierende Format-Provider-Klasse beim Aufbereiten des Strings verzichten und sich ganz auf Formatzeichenfolgen verlassen.

Für unser Beispiel ist der Aufwand umso geringer, als wir eine Formatierungs-Engine, die Formatzeichen auswerten kann, von vornherein implementiert haben.

Damit AppendFormat, das in letzter Instanz auch von Console.WriteLine für das Zusammenbauen einer Parameterzeichenfolge (»text {0} text {1} ...«) verwendet wird, auf diese Formatierungs-Engine unserer Beispielklasse zurückgreift, brauchen wir lediglich die IFormattable-Schnittstelle in unsere Laengen-Klasse einzubinden:

```
Public Class Laengen
    Implements IFormattable

    'Speichert die Länge in Meter.
    Private myLaenge As Decimal

    Sub New(ByVal Meter As Decimal)
        myLaenge = Meter
    End Sub

    .
    .
    .
```

IFormattable verlangt, dass es eine ToString-Funktion mit entsprechender Signatur in der sie einbindenden Klasse gibt. Die Signatur verlangt, dass eine Formatzeichenfolge und ein IFormatProvider übergeben werden können. Aber auch eine solche ToString-Version gibt es in unserer Klasse bereits:

```

Public Overloads Function ToString(ByVal formatChars As String, _
    ByVal formatProvider As System.IFormatProvider) As String Implements IFormattable.ToString

    Trace.WriteLine("ToString (Formattable-Signatur) wurde aufgerufen!")

    If (TypeOf formatProvider Is CultureInfo) Or formatProvider Is Nothing Then
    .
    .
    .

```

Wir waren gut vorbereitet, was? Denn das war es auch schon. Wenn Sie die Formatierungsprozedur

```

Sub FormatterTest()

    'Definiert eine Laengen-Instanz mit 1 (einem Meter).
    Dim locLaenge As New Laengen(1)
    Dim locLaengenFormatInfo As New LaengenFormatInfo

    locLaengenFormatInfo.Aufloesung = LaengenAufloesung.SehrKlein
    locLaengenFormatInfo.Kultur = LaengenKultur.EnglischAmerikanisch

    'Testen der Format-Funktion.
    Dim locStr As String = String.Format(locLaengenFormatInfo, _
        "Testausgabe {0:; #.##0.00} eines Laengen-Objektes", _
        locLaenge)
    Console.WriteLine(locStr)
    'Testen des Formatters.
    Console.WriteLine("Testausgabe {0:g-d; #.##0.00} eines Laengen-Objektes", locLaenge)

    Console.ReadLine()
End Sub

```

anschließend abermals starten, sehen Sie folgendes Ergebnis auf dem Bildschirm:

```

Testausgabe 472,40000 eines Laengen-Objektes
Testausgabe ,00100 eines Laengen-Objektes

```

Sie sehen: Sowohl `String.Format` mit Unterstützung eines eigenen Format Providers als auch `WriteLine` funktionieren jetzt perfekt!

