

Kapitel 28

Steuerelemente

In diesem Kapitel:

Allgemeines	690
Statische Anzeigefelder	718
Schaltflächen	720
Textfelder	725
Drehfelder, Listenfelder und Kombinationsfelder	728
Datumsauswahl	733
Laufleisten	736
Listen- und Strukturansicht	742
Container-Elemente	758
Menü-, Symbol- und Statusleiste	760
Webbrowser	771

Nachdem im Abschnitt zum Windows Forms-Designer aus Kapitel 26 bereits die allgemeinen Grundlagen zur Programmierung mit Steuerelementen behandelt wurden, ist das vorliegende Kapitel Einzelbeschreibungen ausgesuchter Steuerelemente gewidmet. Zuvor aber gehen wir noch auf die von `Control` vererbten, gemeinsamen Eigenschaften und Fähigkeiten der Steuerelemente ein.

Allgemeines

Die folgenden Abschnitte behandeln thematisch geordnet die Eigenschaften und Methoden, die von `Control` an die einzelnen Steuerelement-Klassen vererbt werden und daher grundsätzlich allen Steuerelementen zur Verfügung stehen.

HINWEIS Die Klasse `Control` vererbt eine ganze Reihe von grundlegenden Eigenschaften, die allerdings nicht für alle abgeleiteten Steuerelemente sinnvoll sind. So verfügen z. B. Fortschrittsanzeigen weder über Text noch sonstigen Inhalt, an den die Größe des Steuerelements angepasst werden könnte. Folglich sind die von `Control` geerbten Eigenschaften `Text` und `AutoSize` für `ProgressBar`-Instanzen ohne Bedeutung. Sie sind allerdings trotzdem vorhanden (OOP kennt keine selektive Vererbung). Im Eigenschaftfenster werden Sie sie allerdings nicht sehen, da die abgeleiteten Steuerelemente-Klassen »unsinnige« Eigenschaften mit dem `Browsable`-Attribut verbergen (siehe Ende von Kapitel 29).

Darstellung

Die wichtigsten Eigenschaften für das Erscheinungsbild der Steuerelemente sind zweifelsohne die Vorder- und Hintergrundfarbe (`ForeColor` und `BackColor`) sowie die Schrift (`Font`).

Schrift

Im Eigenschaftfenster des Windows Forms-Designers können Sie die Schrift komfortabel über ein Dialogfeld auswählen. Zur Laufzeit erzeugen Sie Schriften mithilfe des vielfach überladenen `Font`-Konstruktors, beispielsweise durch Angabe des Font-Namens oder durch Abwandlung eines bestehenden `Font`-Objekts:

```
btn.Font = new Font("Times New Roman", 12,  
                  FontStyle.Bold | FontStyle.Underline);  
btn.Font = new Font(myform.Font, FontStyle.Bold);
```

Wenn Sie eine Schriftart verwenden, die auf dem System, auf dem die Anwendung später ausgeführt wird, nicht vorhanden ist, wird auf die Ersatzschrift *Microsoft Sans Serif* zurückgegriffen. Mehr zur Erzeugung und Verwendung von Schriftarten in Kapitel 31 zu GDI+.

HINWEIS `Font`-Objekte sind unveränderlich.

ACHTUNG `Font` gehört zu den Ambient-Eigenschaften, siehe Abschnitt weiter unten.

Farben

Im Eigenschaftsfenster wählen Sie die Vorder- (ForeColor) und Hintergrundfarbe (BackColor) für Ihre Steuerelemente aus einem dreiteiligen Listenfeld aus.

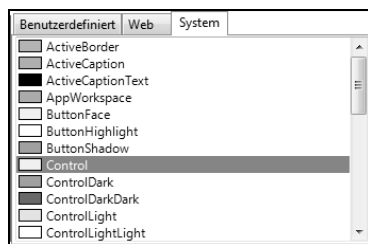


Abbildung 28.1 Das Farben-Listenfeld des Eigenschaftsfensters

Auf der Registerkarte *Benutzerdefiniert* können Sie Farben frei auswählen oder auch aus Rot-, Grün- und Blauanteilen komponieren (klicken Sie dazu mit der rechten Maustaste in eines der leeren Felder unterhalb der vordefinierten Farbkästchen).

Für Steuerelemente sollten Sie aber grundsätzlich die *System*-Farben vorziehen. Dabei wählen Sie keine konkrete Farbe aus, sondern jeweils die Farbe, die auf dem aktuellen System für bestimmte Teile der Benutzeroberfläche vorgesehen ist. Wenn Sie also z. B. für die BackColor-Eigenschaft einer Schaltfläche die System-Farbe Control auswählen, wird Ihre Schaltfläche beim Anwender genau die Hintergrundfarbe zeigen, die der Anwender auf seinem System als Hintergrundfarbe für Steuerelemente eingestellt hat.¹

Zur Laufzeit weisen Sie als Farbe ein System.Drawing.Color- oder System.Drawing.SystemColors-Objekt zu. Oder Sie erzeugen eigene Farben mit der FromArgb()-Methode der Color-Struktur. Mehr hierzu in Kapitel 31 zu GDI+.

```
btn.ForeColor = Color.Red;
btn.BackColor = System.Drawing.SystemColors.Control;
btn.ForeColor = Color.FromArgb(0, 255, 0, 0);
```

ACHTUNG

BackColor und ForeColor gehören zu den Ambient-Eigenschaften, siehe Abschnitt weiter unten.

Name	Beschreibung
ActiveBorder	Rahmenfarbe des aktiven Fensters
ActiveCaption	Hintergrundfarbe der Titelleiste des aktiven Fensters
ActiveCaptionText	Farbe des Textes auf der Titelleiste des aktiven Fensters
AppWorkspace	Farbe des Arbeitsbereichs von MDI-Anwendungen
ButtonFace	Vorderseitenfarbe eines 3D-Elements
ButtonHighlight	Hervorhebungsfarbe eines 3D-Elements
ButtonShadow	Schattenfarbe eines 3D-Elements

¹ Unter Windows XP kann der Anwender die Systemfarben über den Dialog *Eigenschaften von Anzeige*, Registerkarte *Darstellung* festlegen. Aufgerufen wird das Dialogfeld über den Befehl *Eigenschaften* im Kontextmenü des Desktops. Unter Windows Vista bzw. Windows 7 wählen Sie im Kontextmenü den Befehl *Anpassen* und dann entweder den Link *Fensterfarbe und -darstellung* und schließlich die Schaltfläche *Erweitert* (Vista) oder den Link *Fensterfarbe* (7).

Name	Beschreibung
Control	Vorderseitenfarbe eines 3D-Elements
ControlDark	Schattenfarbe eines 3D-Elements
ControlDarkDark	Dunkle Schattenfarbe eines 3D-Elements
ControlLight	Helle Farbe eines 3D-Elements
ControlLightLight	Hervorhebungsfarbe eines 3D-Elements
ControlText	Textfarbe eines 3D-Elements
Desktop	Farbe des Desktops
GradientActiveCaption	Hellste Farbe im Farbverlauf auf einer Titelleiste des aktiven Fensters
GradientInactiveCaption	Hellste Farbe im Farbverlauf auf der Titelleiste eines inaktiven Fensters
GrayText	Farbe von abgeblendetem Text
Highlight	Hintergrundfarbe ausgewählter Elemente
HighlightText	Farbe des Textes ausgewählter Elemente
HotTrack	Farbe, die zum Kennzeichnen eines vorselektierten Elements verwendet wird
InactiveBorder	Rahmenfarbe eines inaktiven Fensters
InactiveCaption	Hintergrundfarbe der Titelleiste eines inaktiven Fensters
InactiveCaptionText	Farbe des Textes auf der Titelleiste eines inaktiven Fensters
Info	Hintergrundfarbe einer QuickInfo
InfoText	Textfarbe einer QuickInfo
Menu	Hintergrundfarbe eines Menüs
MenuBar	Hintergrundfarbe einer Menüleiste
MenuHighlight	Farbe, die zum Hervorheben von Menüelementen verwendet wird, wenn das Menü als flaches Menü dargestellt wird
MenuText	Textfarbe eines Menüs
ScrollBar	Hintergrundfarbe einer Bildlaufleiste
Window	Hintergrundfarbe des Clientbereichs eines Fensters
WindowFrame	Farbe eines Fensterrahmens
WindowText	Farbe des Textes im Clientbereich eines Fenster

Tabelle 28.1 Die Systemfarben von Windows

Hintergrundbilder und Bildlisten

Einige Steuerelemente, darunter z.B. Button, ListView, TreeView und PictureBox, erlauben statt einer Hintergrundfarbe auch die Einblendung eines Hintergrundbildes.

Im Eigenschaftfenster können Sie die Bilddatei des Hintergrundbildes über das zur BackgroundImage-Eigenschaft gehörende Dialogfeld importieren (wahlweise in die lokale Ressourcendatei des Formulars oder die Projektressourcendatei) und gleich an die Eigenschaft zuweisen.

Zur Laufzeit können Sie das Hintergrundbild aus einer Ressourcendatei oder direkt aus einer Datei laden und zuweisen:

```
// Aus Projektressourcendatei laden
btn.BackgroundImage = WindowsApplication.Properties.Resources.HGBild;
// Aus Bilddatei laden
btn.BackgroundImage = new Bitmap(@"..\images\HGBild.jpg");
```

Die meisten Steuerelemente, die Hintergrundbilder unterstützen, verfügen zudem über die Eigenschaft `BackgroundImageLayout`, mit der die Anpassung des Bildes an die Abmessungen des Steuerelements eingestellt werden kann. Mögliche Werte sind:

Konstante	Beschreibung
None	
Center	rt im Steuerelement ausgerichtet (falls das Bild kleiner ist als das Steuerelement)
Stretch	Das Bild wird an die Größe des Steuerelements angepasst
Tile	Das Steuerelement wird mit dem Bild gekachelt (falls das Bild kleiner ist als das Steuerelement)
Zoom	Das Bild wird an die Größe des Steuerelements angepasst, behält aber seine ursprünglichen Proportionen

Werte der Enumeration `ImageLayout`

Wenn Sie für die Gestaltung der Benutzeroberfläche eines Fensters mehrere Bilder benötigen, lohnt es sich unter Umständen, die Bilder in einer `ImageList`-Komponente zu verwalten. Ziehen Sie dazu einfach eine `ImageList`-Instanz aus der Toolbox in Ihr Fenster und öffnen Sie im Eigenschaftfenster den Dialog zur Images-Eigenschaft, um die gewünschten Bilder zu laden.

Zur Laufzeit benutzen Sie die überladene `Add()`-Methode, um Bilder hinzuzufügen, und `Clear()` bzw. `RemoveAt(index)`, um Bilder aus der Komponente zu entfernen.

```
ImageList il = new ImageList();

// Bitmap aus Datei laden (und wahlweise die transparente Farbe angeben)
il.Images.Add( Image.FromFile(@"..\images\HGBild.jpg") );
il.Images.Add( Image.FromFile(@"..\images\HGBild.jpg"), Color.White);

// Icon aus Datei laden
il.Images.Add( new Icon(@"..\images\myicon.ico") );
```

Bitmaps aus einer `ImageList` können Sie nicht über das Eigenschaftfenster an die Eigenschaften der Steuerelemente zuweisen. Der benötigte Code ist allerdings schnell von Hand aufgesetzt, vorausgesetzt Sie kennen den Index des Bildes:

```
// Bild aus ImageList il zuweisen
btn.BackgroundImage = il.Images[0];
```

Titel

Viele Steuerelemente besitzen einen Titel, eine Beschriftung oder anzuzeigenden Text. Diesen weisen Sie der Eigenschaft `Text` zu. Meist ist dann auch die Eigenschaft `TextAlign` verfügbar, mit der Sie die Ausrichtung des Textes im Steuerelement festlegen können.

Wenn Sie über einem einzelnen Steuerelement den Warte-Cursor (Sanduhr) anzeigen möchten, können Sie auch statt den Cursor explizit zu tauschen, einfach die Eigenschaft `UseWaitCursor` auf `true` setzen.

Wenn Sie nicht nur für ein einzelnes Steuerelement, sondern die ganze Anwendung den Cursor wechseln möchten, weisen Sie den gewünschten Cursor der statischen `Current`-Eigenschaft der Klasse `Cursor` zu.

```
// in einer Methode
Cursor.Current = Cursors.WaitCursor;
...
Cursor.Current = Cursors.Default;
```

HINWEIS Zuweisungen an die Eigenschaft `UseWaitCursor` werden als synchrone Nachrichten verarbeitet. Die Eigenschaft eignet sich folglich nicht, um den Cursor innerhalb einer Methode/Ereignisbehandlung zu ändern und wieder zurückzusetzen.

ACHTUNG Cursor gehört zu den Ambient-Eigenschaften, siehe Abschnitt weiter unten.

Sichtbarkeit und Aktivierung

Die booleschen Eigenschaften `Enabled` und `Visible` sind standardmäßig auf `true` gesetzt und steuern Aktivierung und Sichtbarkeit der Steuerelemente.

Wenn Sie `Enabled` für ein Steuerelement auf `false` setzen, wird das Element deaktiviert, d.h., es wird grau dargestellt und reagiert nicht mehr auf Benutzereingaben – eine elegante Möglichkeit, die Funktionalität von Formularen vorübergehend einzuschränken, ohne die Funktionalität ganz vor dem Anwender verbergen zu müssen. Typische Beispiele sind die Deaktivierung der Befehle *Bearbeiten/Ausschneiden* und *Bearbeiten/Kopieren*, wenn kein Text (oder andere Inhalte) ausgewählt ist (siehe Kapitel 30, »Benutzeroberfläche«), sowie die Deaktivierung der *Weiter*-Schaltfläche von Dialogen, bis diese komplett bearbeitet sind.

Einen Schritt weiter als die Aktivierung geht die Ein- und Ausblendung von Steuerelementen über die Eigenschaft `Visible`. Sie ist allerdings bei weitem nicht so gebräuchlich, da Steuerelemente eher selten ein- und ausgeblendet werden und für Formulare traditionell eher die Methoden `Show()` und `Hide()` – entspricht `Visible = true;` bzw. `Visible = false;` – verwendet werden.

Visuelle Stile

Seit Windows XP kann der Windows-Benutzer das Erscheinungsbild seines Desktops und der laufenden Anwendungen durch Auswahl verschiedener Designs nach seinen Wünschen anpassen. Windows XP-Anwender wählen dazu im Kontextmenü ihres Desktops den Befehl *Eigenschaften* aus und gehen zur Registerkarte *Designs*. Windows Vista-Anwender wählen im Kontextmenü den Befehl *Anpassen* aus und klicken im Dialogfeld auf *Designs*. Windows-7-Anwender können das Design auf der Seite *Anpassung* auswählen (Aufruf über Befehl *Anpassen* im Kontextmenü des Desktops).



Abbildung 28.3 Windows-Designs: klassisch, Windows XP und Windows Vista/7

Während das Erscheinungsbild des Fensters, seiner Titelleiste und der Bildlaufleiste unter Windows XP/Vista automatisch an das vom Benutzer gewählte Design angepasst wird, gilt dies nicht für die Steuerelemente. Damit auch diese sich korrekt dem gewählten Design anpassen und gegebenenfalls auch mit den Designs verbundene stilistische Neuerungen wie abgerundete Ecken oder graduelle Schattierungen zeigen, müssen Sie in der `Main()`-Methode Ihrer Anwendung zuerst die visuellen Stile aktivieren:

```
static void Main()
{
    Application.EnableVisualStyles();
    ...
}
```

Über die Eigenschaft `VisualStyleState` können Sie zudem festlegen, für welche Fensterbereiche das Design übernommen werden soll. Mögliche Werte sind `ClientAndNonClientAreasEnabled` (Standard), `ClientAreaEnabled`, `NonClientAreaEnabled` und `NoneEnabled` der Enumeration `VisualStyleState`.

```
Application.VisualStyleState =
    System.Windows.Forms.VisualStyles.VisualStudioState.ClientAreaEnabled;
```

HINWEIS Bei manchen Steuerelementen, zum Glück aber nur bei wenigen, kann es in bestimmten Konfigurationen zu Darstellungsfehlern kommen. So z.B. bei `MonthCalendar`-Steuerelementen, für die ein Auswahlbereich festgelegt wurde, oder `TabControl`-Elementen mit Registerkarten, die am unteren, rechten oder linken Rand ausgerichtet sind.

Seien Sie nicht enttäuscht, wenn sich die Benutzeroberfläche Ihrer Anwendung trotz eingeschalteter Unterstützung für visuelle Stile nicht dramatisch ändert. Dies liegt dann wahrscheinlich daran, dass Sie vornehmlich Steuerelemente verwenden, die unter allen Designs weitgehend (Kontrollkästchen, Optionsfelder ...) oder vollkommen identisch (`Label`, `LinkLabel`, `NumericUpDown` ...) dargestellt werden.

Bei Schaltflächen sollten Sie allerdings einen Blick auf den Wert der `FlatStyle`-Eigenschaft werfen. Diese Eigenschaft, die im Übrigen nicht von `Control` stammt, beeinflusst Darstellung und Drücken-Animation der Schaltflächen. Der Effekt ist bei Standardschaltflächen am größten, lässt sich aber auch bei Optionsfeldern und Kontrollkästchen beobachten.

Wert	Beschreibung
Flat	Flache Darstellung. Wird der Mauszeiger über die Schaltfläche bewegt, wird das Steuerelement grau. In dieser Darstellung werden Designs/visuelle Stile nicht berücksichtigt.
Popup	Flache Darstellung. Wird der Mauszeiger über die Schaltfläche bewegt, wird das Steuerelement dreidimensional erhaben dargestellt. In dieser Darstellung werden Designs/visuelle Stile nicht berücksichtigt.
Standard	Dreidimensionale Darstellung. Berücksichtigt Designs/visuelle Stile.
System	Dreidimensionale Darstellung. Das Rendern wird in diesem Fall vom Betriebssystem bestimmt, wodurch bestimmte Einstellungen in den Eigenschaften des Steuerelements ausgehebelt werden können (z. B.: BackColor, TextAlign, Image und ImageAlign). Berücksichtigt Designs/visuelle Stile.

Tabelle 28.4 FlatStyle-Werte für Schaltflächen



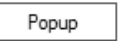

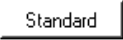


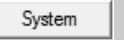
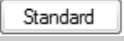
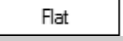

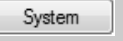
Design	Stile aktiviert?	Darstellung
Windows klassisch	ja/nein	   
Windows Vista/7	nein	   
Windows Vista/7	ja	   

Tabelle 28.5 Darstellung von Schaltflächen unter Windows Vista bei unterschiedlichen Einstellungen für Design (Auswahl auf Betriebssystemebene) und visuelle Stile (Aktivierung auf Programmebene)

Ambient-Eigenschaften

Die Eigenschaften Cursor, Font, BackColor, ForeColor und RightToLeft sind Ambient-Eigenschaften. Ambient-Eigenschaften besitzen keine eigenen Standardwerte, sondern übernehmen per Voreinstellung ihre Werte von dem übergeordneten Container/Formular. Wenn Sie also z. B. mehrere Label in ein Formular einbetten und anschließend die Hintergrundfarbe des Formulars von SystemColors.Control in SystemColors.Window ändern – wobei es keine Rolle spielt, ob Sie die Änderung zur Entwurfszeit im Eigenschaftenfenster oder zur Laufzeit im Code vornehmen –, übernehmen die Labels automatisch die neue Hintergrundfarbe. Sollen die eingebetteten Steuerelemente andere Farben verwenden bzw. Farbänderungen des übergeordneten Formulars nicht mitmachen, müssen Sie den BackColor-Eigenschaften der Steuerelemente explizit die gewünschten Werte zuweisen.

Größe, Position und Ausrichtung

Größe und Position eines Steuerelements werden Sie in der Regel durch Schieben und Ziehen mit der Maus im Windows Forms-Designer festlegen.

Beim Ausrichten helfen die Eigenschaften Anchor und Dock, aber auch die Optionen im Designer-Menü *Format/Ausrichten*).

All das dürfte Ihnen allerdings bereits aus Kapitel 26, Abschnitte »Benutzeroberfläche« und »Windows Forms-Designer« bekannt sein.

QuickInfos

QuickInfo-Texte für Steuerelemente werden in Windows Forms nicht in den einzelnen Steuerelementen, sondern zentral in einer Instanz der Klasse `ToolTip` gespeichert:

```
ToolTip quickinfos = new ToolTip();
...
quickinfos.SetToolTip(btn1, "Dialog verlassen und Eingaben übernehmen");
quickinfos.SetToolTip(btn2, "Dialog verlassen und Eingaben verwerfen");
```

Lediglich für ausgesuchte Oberflächenelemente wie die Schaltflächen von Symbolleisten oder die Knoten einer Baumansicht gibt es inhärente QuickInfos und eine passende Eigenschaft `ToolTipText` zum Setzen des QuickInfo-Textes.


Der Umweg über `ToolTip`-Instanzen hat allerdings nicht nur den Vorteil, dass die QuickInfo-Texte zentral verwaltet werden. Die `ToolTip`-Instanzen erlauben uns zudem die Konfiguration der QuickInfos.



Eigenschaft	Beschreibung
<code>AutomaticDelay</code> <code>InitialDelay</code> <code>ReshowDelay</code>	Steuert die Verzögerung, mit der die QuickInfos angezeigt werden.
<code>BackColor</code> <code>ForeColor</code>	Hinter- und Vordergrundfarbe der QuickInfo-Fenster
<code>IsBalloon</code>	Legt fest, ob ein rechteckiges (<code>false</code>) oder ein Sprachblasen-Fenster (<code>true</code>) verwendet wird.
<code>ShowAlways</code>	Legt fest, ob das QuickInfo-Fenster auch eingeblendet werden soll, wenn das übergeordnete Fenster inaktiv ist (Standardwert <code>false</code>).
<code>ToolTipIcon</code>	Blendet neben dem QuickInfo-Text ein Symbol ein. Mögliche Werte sind: <code>ToolTipIcon.Error</code> , <code>ToolTipIcon.Info</code> , <code>ToolTipIcon.Warning</code> und <code>ToolTipIcon.None</code> .
<code>UseAnimation</code> <code>UseFading</code>	Schaltet Animations- bzw. Ein- und Ausblendeeffekte ein oder aus.

Tabelle 28.6 Konfigurationseigenschaften der `ToolTip`-Klasse

Fokus, Aktivierreihenfolge und Schnellzugriffstasten

Grundsätzlich können alle Steuerelemente, die Benutzereingaben oder -aktionen erwarten, auch über die Tastatur bedient werden. Einige Steuerelemente wie z.B. die Textfelder (`TextBox`, `RichTextBox` ...) sind sogar speziell für den Empfang von Tastatureingaben konzipiert. Voraussetzung dafür, dass ein Steuerelement auf Tastatureingaben reagiert, ist allerdings, dass es den Eingabefokus besitzt.

Der Anwender kann, um einem Steuerelement den Fokus zuzuweisen, das Element anklicken oder die -Taste drücken, bis der Fokus zu dem gewünschten Element gesprungen ist.² All dies geschieht automatisch, ohne dass Sie dafür Code schreiben müssen. Was also bleibt zu tun? Warum sollten Sie sich überhaupt noch mit dem Fokus-Mechanismus beschäftigen? Nun, vor allem können Sie viel dafür tun, die Bedienbarkeit Ihrer Anwendung zu erhöhen, indem Sie

- die Reihenfolge der -Aktivierung optimal einstellen,
- als Alternative zur -Aktivierung Schnellzugriffstasten anbieten,
- wo sinnvoll, einem bestimmten Steuerelement den Fokus direkt per Code zuweisen.

Aktivierreihenfolge



Chaotische, unlogische oder einfach nur unpraktische Aktivierreihenfolgen strapazieren das Nervenkostüm tastaturorientierter Anwender. In Fenstern und Dialogfeldern, die mit Steuerelementen bestückt sind, sollte der Programmierer daher auf jeden Fall kontrollieren, in welcher Reihenfolge die Elemente beim Drücken der -Taste angesteuert werden.





Abbildung 28.4 Formular mit eingblendeter Aktivierreihenfolge (im Windows Forms-Designer)

In Visual Studio laden Sie dazu das Formular in den Windows Forms-Designer, achten darauf, dass das Formular und nicht ein Steuerelement ausgewählt ist, und rufen anschließend den Befehl *Ansicht/Aktivierreihenfolge* auf. Der Designer nummeriert daraufhin die Steuerelemente entsprechend der Aktivierreihenfolge durch (siehe Abbildung 28.4), genauer gesagt:

- Er blendet zu jedem Steuerelement den Wert der Eigenschaft `TabIndex` ein, welcher die Position des Elements in der Aktivierreihenfolge bestimmt
- Steuerelemente, die einem zwischengeordneten Container-Element angehören, erhalten die Nummer des Containers und eine weitere Nummer, die die Position der Aktivierung mit den Pfeiltasten angibt

Um die Aktivierung neu zu ordnen, brauchen Sie dann nur die Steuerelemente nacheinander in der Reihenfolge, in der sie später aktiviert werden sollen, anzuklicken. Oder Sie beenden die Anzeige durch Drücken der -Taste und weisen einzelnen Steuerelementen die gewünschten `TabIndex`-Werte im Eigenschaftsfenster zu.

² Steuerelemente, die in Container-Elemente wie `Panel` oder `GroupBox` eingebettet sind, können üblicherweise mit den Pfeiltasten erreicht werden. D. h. der Anwender springt mit der -Taste zu dem ersten Element im Container und dann mit den Pfeiltasten weiter zu dem gewünschten Element.

Wenn Sie ein bestimmtes Steuerelement von der -Aktivierung ausnehmen möchten, setzen Sie seine `TabStop`-Eigenschaft auf `false`. Wenn Sie seine `Enabled`-Eigenschaft auf `false` setzen, wird das Steuerelement vollständig deaktiviert.

HINWEIS Einige Steuerelemente wie `Label` oder `ProgressBar` können den Eingabefokus nicht übernehmen. Ihre `TabStop`-Eigenschaft ist auf `false` gesetzt und kann nicht geändert werden (im Eigenschaftenfenster wird sie gar nicht erst angezeigt).

Schnellzugriffstasten

Je mehr Steuerelemente ein Fenster oder Dialogfeld enthält, umso enervierender ist es, ein bestimmtes Element über die Aktivierreihenfolge ansteuern zu müssen. Natürlich bleibt immer die Alternative zur Maus zu greifen; bedienerfreundlicher sind allerdings Programme, die für die Steuerelemente zusätzlich Schnellzugriffstasten definieren.

Dazu müssen Sie lediglich in der `Text`-Eigenschaft des Steuerelements dem für den Schnellzugriff zu verwendenden Buchstaben ein kaufmännisches Und (`&`) voranstellen.

```
btn1.Text = "&Weitersuchen";
```

Steuerelemente, die keine `Text`-Eigenschaft haben, stellen Sie einfach ein `Label`-Element mit Schnellzugriff zur Seite. Wenn dieses in der Aktivierreihenfolge direkt vor dem anzusteuernenden Steuerelement liegt, springt bei Drücken der Schnellzugriffstaste der Fokus direkt zu dem nachgeschalteten Steuerelement.

HINWEIS Wenn Sie in der `Text`-Eigenschaft eines Steuerelements das `&`-Zeichen verwenden möchten, ohne dass dieses die Schnellzugriffstaste kennzeichnen soll, müssen Sie die `UseMnemonic`-Eigenschaft des Steuerelements auf `false` setzen.

Fokus setzen

Gelegentlich werden Sie in die Verlegenheit kommen, den Fokus vom Code aus auf ein bestimmtes Steuerelement setzen zu müssen.

Rufen Sie dazu einfach die `Select()`-Methode des Steuerelements auf oder weisen Sie den Verweis auf das Steuerelement der `ActiveControl`-Eigenschaft des Formulars zu.

```
btnOK.Select();
```

Beachten Sie, dass die Zuweisung des Fokus nicht immer von Erfolg gekrönt sein muss. Abgesehen davon, dass bestimmte Steuerelemente den Fokus grundsätzlich nicht übernehmen können (siehe Hinweis), ist auch die Zuweisung an deaktivierte (`Enabled` auf `false`) oder nicht sichtbare (`Visible` auf `false`) Steuerelemente nicht möglich. Wenn Sie auf eine gescheiterte Fokus-Zuweisung reagieren möchten, fragen Sie zuerst den Rückgabetypp der Eigenschaft `CanFocus` ab:

```
if (btn.CanFocus)
{
    btn.Select();
}
else
```

```
{  
    // alternativer Code  
}
```

Um festzulegen, welches Steuerelement bei Erzeugung eines Formulars den Fokus erhält, genügt es in der Regel, einen Blick auf die Aktivierreihenfolge zu werfen. Standardmäßig wird nämlich dem ersten Element in der Aktivierreihenfolge der Fokus zugewiesen und meist entspricht dies genau der gewünschten Einstellung.

Wird allerdings das Formular mehrfach ein- und ausgeblendet (`Show()` und `Hide()`), behält das jeweils zuletzt fokussierte Steuerelement im Formular den Fokus bei. Ist dieses Verhalten unerwünscht, so können Sie das Ereignis `Shown` behandeln und in der zugehörigen Behandlungsmethode den Anfangsfokus explizit setzen:

```
private void OptionDialog_Shown(object sender, EventArgs e)  
{  
    btnOK.Select();  
}
```

HINWEIS `GroupBox`, `Label`, `LinkLabel`, `Panel`, `PictureBox`, `ProgressBar` und `Splitter` kann der Fokus nicht zugewiesen werden.

ACHTUNG Setzen Sie den Fokus nicht innerhalb einer `LostFocus`-Ereignisbehandlungsmethode!

Auswertung

In Windows Forms sind Eingabe-Steuerelemente grundsätzlich so ausgelegt, dass sie die vom Anwender vorgesehenen Eingaben/Einstellungen in speziellen, Steuerelement-spezifischen Eigenschaften speichern. So speichern `TextBox`-Felder den eingegebenen Text in der `Text`-Eigenschaft und Kontrollkästchen speichern ihren Zustand in der `Checked`-Eigenschaft. Der Programmierer braucht also nichts weiter zu tun, als die betreffenden Eigenschaften zum richtigen Zeitpunkt abzufragen und auszuwerten. Doch wann ist der richtige Zeitpunkt?

Auswertung während der Eingabe

Wenn Sie bereits reagieren möchten, während der Anwender noch beim Bearbeiten des Steuerelements ist, so sollten Sie nach den `Changed`-Ereignissen des Steuerelements Ausschau halten.

Auch wenn der Anwender die Eingaben im Steuerelement zu diesem Zeitpunkt noch nicht als »endgültig« abgesegnet hat (indem er das Steuerelement verlassen oder den zugehörigen Dialog durch Drücken der OK-Schaltfläche beendet hat), können Sie die `Changed`-Ereignisse nutzen, um Live-Effekte zu erzielen (der Anwender sieht sofort die Auswirkung seiner Einstellung) oder um die Eingabe selbst zu überwachen und notfalls zu korrigieren.

Letzteres demonstriert der folgende Code, der Umlaute, die in ein `TextBox`-Feld eingegeben werden, automatisch in Umschreibungen ändert.

```

private void textBox1_TextChanged(object sender, EventArgs e)
{
    string s = textBox1.Text;

    if (!String.IsNullOrEmpty(s))
    {
        switch(s[s.Length - 1])
        {
            case 'ä': s = s.Substring(0, s.Length - 1);
                     s = s + "ae";
                     break;
            case 'Ä': s = s.Substring(0, s.Length - 1);
                     s = s + "Ae";
                     break;
            case 'ö': s = s.Substring(0, s.Length - 1);
                     s = s + "oe";
                     break;
            case 'Ö': s = s.Substring(0, s.Length - 1);
                     s = s + "Oe";
                     break;
            case 'ü': s = s.Substring(0, s.Length - 1);
                     s = s + "ue";
                     break;
            case 'Ü': s = s.Substring(0, s.Length - 1);
                     s = s + "Ue";
                     break;
            case 'ß': s = s.Substring(0, s.Length - 1);
                     s = s + "ss";
                     break;
        }

        textBox1.Text = s;
        textBox1.SelectionStart = textBox1.Text.Length;
    }
}

```

Auswertung direkt nach der Eingabe

Der günstigste Zeitpunkt, um Eingaben oder Einstellungen *einzelner* Steuerelemente direkt auszuwerten, ist, wenn der Anwender das Steuerelement verlässt, sprich der Fokus weiterwandert. In diesem Moment wird allerdings gleich eine ganze Kaskade von Ereignissen ausgelöst: Leave, LostFocus, Validating und Validated.

- Leave ist für eine einfache Auswertung und Reaktion auf die Benutzereingabe ideal. Wenn Sie hingegen zuerst die Korrektheit der Eingaben überprüfen wollen, behandeln Sie die Ereignisse.
- Validating (zum Prüfen der Korrektheit) und Validated (Auswertung, sofern gültige Eingaben vorliegen). Setzen Sie in der Behandlungsmethode des Validating-Ereignisses die Cancel-Eigenschaft des CancelEventArgs-Parameters auf true, um die weiteren Ereignisse zu unterbinden und den Fokuswechsel zu verhindern.

```

private void textBox1_Validating(object sender, CancelEventArgs e)
{
    if (String.IsNullOrEmpty(textBox1.Text))
        e.Cancel = true;
}


private void textBox1_Validated(object sender, EventArgs e)
{
    textBox2.Text = textBox1.Text;
}

```

- `LostFocus` werden Sie, falls Sie mit Visual Studio arbeiten, im Eigenschaftfenster allerdings umsonst suchen. Es gilt zusammen mit `GotFocus` als Lowlevel-Ereignis und ist für die Implementierung eigener Steuerelemente gedacht. Trotzdem könnte es in Einzelfällen interessant sein, da es im Gegensatz zu den anderen Ereignissen auch dann ausgelöst wird, wenn der Anwender das Fenster wechselt! Den Code zur Einrichtung der Ereignisbehandlungsmethode müssen Sie dann allerdings selbst aufsetzen:

```
textBox1.LostFocus += textBox1_LostFocus;
...
private void textBox1_LostFocus(object sender, EventArgs e)
{
    // Ereignis behandeln
}
```

HINWEIS Wenn Sie für ein Steuerelement die Eigenschaft `CausesValidation` auf `false` setzen, löst das Element die Ereignisse `Validating` und `Validated` nicht aus. Die Ereignisse werden auch nicht für das Element ausgelöst, von dem der Fokus übernommen wurde. Die `Validating`- und `Validated`-Ereignisse dieses Elements werden erst dann ausgelöst, wenn der Fokus an ein Element geht, dessen `CausesValidation`-Eigenschaft auf `true` gesetzt ist.

ACHTUNG Wenn Sie das `LostFocus`-Ereignis abfangen, müssen Sie nicht nur beachten, dass dieses Ereignis auch dann ausgelöst wird, wenn der Anwender den Fokus auf dem Steuerelement belässt, aber ein anderes Fenster aktiviert. Außerdem ist zu berücksichtigen, dass das Ereignis in Abhängigkeit von der Art der Fokus-Weitergabe zu unterschiedlichen Zeitpunkten ausgelöst wird. Bei Fokusweitergabe mit der -Taste, der `Select()`-Methode oder der `ActiveControl`-Eigenschaft wird das Ereignis erst ausgelöst, nachdem bereits das `Enter`-Ereignis des neu anvisierten Steuerelements ausgelöst wurde (siehe Tabelle 28.7).

Weitergabe mit Tab, <code>Select()</code> , <code>ActiveControl</code>	Weitergabe mit Maus, <code>Focus()</code>
Leave A	LostFocus A
Validating A	Leave A
Validated A	Validating A
Enter B	Validated A
LostFocus A	Enter B
GotFocus B	GotFocus B

Tabelle 28.7 Ereignisse bei Fokusweitergabe von Steuerelement A zu B

Gesammelte Auswertung auf Knopfdruck

Die Eingaben in Dialogsteuerelementen werden üblicherweise erst dann ausgewertet, wenn der Anwender durch Drücken entsprechender Schaltflächen anzeigt, dass er mit der Eingabe fertig ist und die Eingabe ausgewertet haben möchte (OK-Schaltfläche modaler Dialoge, *Weitersuchen*-Schaltfläche nicht modaler Suchdialoge etc.). Beispiele hierfür finden Sie in Kapitel 27, Abschnitt »Dialogfelder«.

ErrorProvider als Ausfüllhilfe

Die meisten Eingabe-Steuerelemente (und auf höherer Ebene die Eingabe-Dialoge) sind relativ nachsichtig gegenüber dem Anwender, d.h., sie akzeptieren jegliche Eingabe und wenn sie nicht bearbeitet werden, ist dies in der Regel auch nicht tragisch. In anderen Fällen sind korrekte Eingaben aber unbedingt notwendig. Dann muss dem Anwender irgendwie angezeigt werden, welche Eingaben von ihm erwartet werden und wie diese korrekt zu lauten haben. Aus den vielen denkbaren Lösungsansätzen für dieses Problem möchte ich Ihnen zwei kurz vorstellen.

Unter Dialogfeldern weit verbreitet ist Variante 1, bei der einfach die Schaltfläche zum Auswerten und Verarbeiten des Dialogs so lange deaktiviert wird, bis die Steuerelemente im Dialog korrekt ausgefüllt sind. Man kennt dies beispielsweise von Lizenzabfragen oder Suchdialogen, deren *Weitersuchen*-Schaltfläche deaktiviert wird, wenn kein Suchbegriff eingetragen ist.

Die zweite Variante beruht auf der *ErrorProvider*-Komponente aus dem *System.Windows.Forms*-Namespace. Sie erlaubt es dem Programmierer, falsch ausgefüllte Steuerelemente auf einfachste Weise zu markieren und mit Hinweisen zur korrekten Bearbeitung zu versehen. Alles, was Sie tun müssen, ist:

1. Eine *ErrorProvider*-Instanz aus der Toolbox in das Formular zu ziehen. Wenn Sie nicht mit Visual Studio arbeiten, definieren Sie in dem Formular ein Feld vom Typ *ErrorProvider* und erzeugen Sie die zugehörige Instanz im Konstruktor.
2. An geeigneter Stelle – beispielsweise in der *Click*-Behandlungsmethode der *OK*-Schaltfläche des Dialogs oder der *Validating*-Methode der einzelnen Steuerelemente – die Eingaben in dem Steuerelement zu prüfen und bei Unkorrektheiten die *SetError()*-Methode der *ErrorProvider*-Instanz aufzurufen, mit einem Verweis auf das Steuerelement und einem passenden Hinweistext als Aufrufargumente
3. An geeigneter Stelle durch Aufruf der *Clear()*-Methode der *ErrorProvider*-Instanz alle veralteten Ausfüllhilfen löschen.

Der nachfolgende Code demonstriert die Vorgehensweise am Beispiel eines Textfelds, in das der Anwender Vor- und Nachnamen eingeben soll.

```
public Form1()
{
    // ...
    errprov = new ErrorProvider();
}
private void textBox1_Validating(object sender, CancelEventArgs e)
{
    // ...

    // alte Meldungen löschen
    errprov.Clear();

    // positionieren
    errprov.SetIconPadding(textBox1,5);
    errprov.SetIconAlignment(textBox1, ErrorIconAlignment.BottomLeft);

    // ggf. Fehlermeldung anzeigen
    if (textBox1.Text.IndexOf(' ') == -1)
    {
```



```
errprov.SetError(textBox1, "Bitte Vor- und Nachnamen angeben!");  
e.Cancel = true;  
}  
}
```

Hier wird in der Validating-Behandlungsmethode eines Textfelds überprüft, ob die Eingabe ein Leerzeichen enthält. Ist dies nicht der Fall, wird das Steuerelement mithilfe der ErrorProvider-Instanz errprov markiert (Abbildung 28.5). Fährt der Anwender anschließend mit der Maus über das Symbol mit dem Ausrufezeichen, wird der an SetError() übergebene Text als QuickInfo eingeblendet.



Abbildung 28.5 Textfeld mit ErrorProvider-Meldung

Mittels der Methoden SetIconAlignment() und SetIconPadding() können Sie die Position des Symbols relativ zu den seitlichen Rändern des Steuerelements festlegen. Beide Methoden erwarten als erstes Argument den Verweis auf das Steuerelement.

Als zweites Argument erwartet SetIconAlignment() eine der Konstanten der Enumeration ErrorIconAlignment: BottomLeft, BottomRight, MiddleLeft, MiddleRight, TopLeft, TopRight.

Die Methode SetIconPadding() übernimmt als zweites Argument einen Integer-Wert, der die Anzahl Pixel zwischen Symbol und Steuerelement angibt.

Datenbindung

In der Datenbankprogrammierung werden Steuerelemente vor allem zum Aufbau von Masken benutzt, über die der Anwender die Daten aus der Datenbank betrachten und gegebenenfalls auch ändern kann. Aus dieser speziellen Form der Verwendung von Steuerelementen hat sich das Konzept der Datenbindung entwickelt, mit dem Ziel, den Datenaustausch zwischen Steuerelementen und Datenquellen zu automatisieren und den Programmieraufwand zu reduzieren. Glücklicherweise kommen als Datenquellen dabei nicht nur Datenbanken in Frage, sodass auch »gewöhnliche« Programme, wo sinnvoll, von dem Konzept profitieren können.

HINWEIS In der Beispielsammlung zu diesem Buch finden Sie ein Projekt *Datenbindung*, in dem die nachfolgend erläuterten Techniken demonstriert werden.

HINWEIS Die Datenbindung in Datenbankanwendungen wird in Kapitel 41 behandelt.

Das Grundprinzip

Die Grundidee der Datenbindung ist es, ein Steuerelement so mit einem bestimmten Datenelement zu verbinden, dass Änderungen am Wert des Datenelements sofort im Steuerelement zu sehen sind und umgekehrt Änderungen an dem im Steuerelement angezeigten Wert sofort zurück in das Datenelement geschrieben werden.

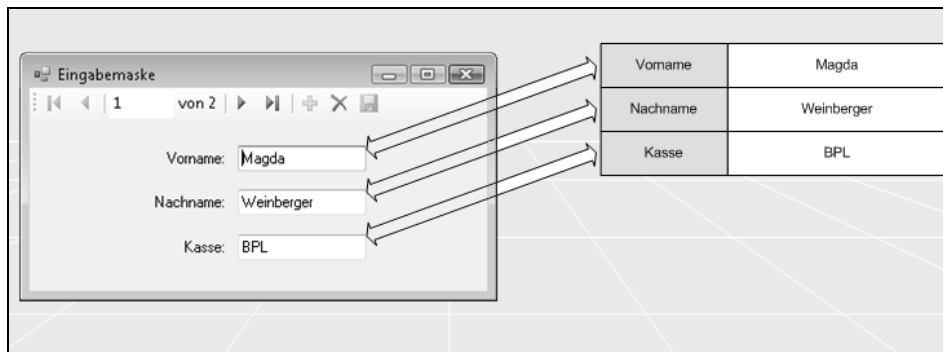


Abbildung 28.6 Datenbindungen sind wie Kanäle, über die die Inhalte von Steuerelementen und die verbundenen Datenquellen in einer oder beiden Richtungen synchronisiert werden

Verbunden wird jeweils die Eigenschaft eines Steuerelements (Ziel) mit der Eigenschaft eines Objekts oder der Eigenschaft des aktuellen Objekts in einer Liste von Objekten (Quelle). Doppeldeutigkeiten sind nicht erlaubt, d.h. ein Ziel kann weder mit mehreren Quellen, noch kann eine Quelle mit mehreren Zielen verbunden werden.

Je nach Beschaffenheit der übertragenen Daten wird zwischen einfachen und komplexen Datenbindungen unterschieden:

- *Einfache Datenbindung* liegt vor, wenn immer nur ein einzelner Wert von der Datenquelle zum Anzeige-Steuerelement übertragen wird.
- Von *komplexer Datenbindung* sprechen wir dagegen, wenn das Steuerelement über die Bindung mehrere Daten von der Quelle entgegen nimmt und anzeigt. Die komplexe Datenbindung ist nur zwischen bestimmten, dafür eingerichteten Steuerelementen (ListBox, DataGridView etc.) und IList-Datenquellen möglich.

Für jede Datenbindung können Sie einstellen, wie die Daten bei der Synchronisierung formatiert werden sollen und wann genau die Synchronisierung stattfinden soll. Zum Eingriff in die Formatierung gibt es die Ereignisse `Format` und `Parse`; die Synchronisierung steuern Sie über die Werte der Enumerationen `ControlUpdateMode` und `DataSourceUpdateMode`. Beide Techniken werden im nachfolgenden Abschnitt dargestellt.

Einfache Datenbindungen zwischen Steuerelement-Eigenschaften

Datenbindungen werden in Windows Forms durch Objekte der Klasse `Binding` repräsentiert. Dem Konstruktor der Klasse werden dabei der Name der Zieleigenschaft, ein Verweis auf das Datenquellen-Objekt und der Name der Quelleigenschaft übergeben. Zur Komplettierung der Bindung fehlt also noch das Zielobjekt. Dies ist immer das aktuelle Objekt, dem Sie die Bindung zuweisen. Die Steuerelemente verfügen zu diesem Zweck über eine Eigenschaft `DataBindings`, hinter der sich eine Auflistung von Bindungen verbirgt. Mithilfe der `Add()`-Methode fügen Sie die gewünschten Bindungen für die Eigenschaften hinzu und schließen auf diese Weise – um es bildlich auszudrücken – den Bindungskanal an das Steuerelement an.

```
Binding b = new Binding("Text", textbox1, "Text");
label1.DataBindings.Add(b);
label2.DataBindings.Add(b);
```

Hier wird zuerst eine Bindung zwischen der Text-Eigenschaft eines TextBox-Textfelds und der Text-Eigenschaft eines beliebigen Steuerelements konfiguriert. Diese wird dann zwei Label-Elementen hinzugefügt, sodass die Eingaben in dem Textfeld automatisch in den Labels widergespiegelt werden.

Wenn Sie die Datenquelle mit nur einem Steuerelement verbinden wollen (und keine eigenen Formatierungsmethoden vorgeben müssen), können Sie das Bindungsobjekt auch implizit von der Add()-Methode erstellen lassen:

```
label1.DataBindings.Add("Text", textbox1, "Text");
```

HINWEIS Vielleicht fragen Sie sich, wie die Datenbindung überhaupt funktionieren kann, wenn Quell- und Zieleigenschaft nur als Strings angegeben werden? Die Lösung ist, dass aus den Strings mittels Reflection³ echte Eigenschaften-Zugriffe generiert werden.

Synchronisierung

Die Synchronisierung läuft grundsätzlich immer in beide Richtungen, also von der Datenquelle zum Steuerelement und vom Steuerelement zur Datenquelle. Sie können allerdings für beide Richtungen individuell einstellen, ob und wann genau synchronisiert werden soll. Die Bindung definiert dazu die beiden Eigenschaften `ControlUpdateMode` und `DataSourceUpdateMode`.

Über die Eigenschaft `ControlUpdateMode` kontrollieren Sie, wann das Steuerelement mit dem Wert aus der Datenquelle aktualisiert wird. Mögliche Werte sind `Never` und `OnPropertyChanged`. Letzterer aktualisiert das Steuerelement jedes Mal, wenn sich der Wert der Quelleigenschaft ändert, genauer gesagt, wenn deren *EigenschaftnameChanged*-Ereignis ausgelöst wird. Der Standardwert ist `OnPropertyChanged`.

Über die Eigenschaft `DataSourceUpdateMode` kontrollieren Sie, wann die Datenquelle mit dem Wert aus dem Steuerelement aktualisiert wird. Mögliche Werte sind:

- `Never` – niemals
- `OnValidation` – bei Validierung, d.h. wenn das Steuerelement dabei ist, den Fokus zu verlieren, und sein *Validating*-Ereignis ausgelöst wird
- `OnPropertyChanged` – jedes Mal, wenn sich der Wert der Steuerelementeigenschaft ändert, d.h., wenn das *EigenschaftnameChanged*-Ereignis ausgelöst wird

Der Standardwert ist `OnValidation`.

```
Binding b = new Binding("Text", obj, "Text");  
b.ControlUpdateMode = ControlUpdateMode.Never;  
b.DataSourceUpdateMode = DataSourceUpdateMode.OnPropertyChanged;
```

³ Reflection ist eine Technologie, mittels der ein Programm »unbekannte« Objekte zur Laufzeit analysieren, erzeugen und steuern kann. Zentrales Element dieser Technologie ist dabei die Identifizierung von Klassenelementen über ihre Namen. So fragt beispielsweise der folgende Code den Wert einer Eigenschaft `Text` ab: `obj.GetType().GetProperty("Text").GetValue(obj, null)`. Reflection wird vornehmlich von Komponentencontainern eingesetzt, die mit beliebigen, unbekannten Komponenten arbeiten müssen – wie z.B. der Windows Forms-Designer, der das Ergebnis seiner Komponentenanalyse im Eigenschaftenfenster anzeigt.

Der Wert für die Aktualisierung der Datenquelle kann auch direkt an die `Add()`-Methode der `DataBindings`-Eigenschaft übergeben werden. Sie müssen allerdings zuvor noch einen booleschen Wert für die Formatierung (siehe unten) übergeben:

```
// Änderungen in Steuerelement textbox2 direkt in die Datenquelle textbox1
// übertragen (ohne zu warten, bis textbox2 den Fokus verliert)
textbox2.DataBindings.Add("Text", textbox1, "Text",
    true, DataSourceUpdateMode.OnPropertyChanged);
```

ACHTUNG Die Synchronisierung findet nur statt, wenn die zugehörigen Ereignisse existieren und ausgelöst werden. Wenn Sie also z. B. als Datenquelle eine Eigenschaft wählen, zu der es kein `Changed`-Ereignis gibt (beispielsweise `textbox1.Multiline`), versagt die `ControlUpdateMode.OnPropertyChanged`-Synchronisierung. Und wenn Sie als Steuerelement ein Label verwenden und darauf hoffen, dass Änderungen, die Sie per Code am Text des Labels vornehmen, auch in die verbundene Datenquelle, beispielsweise ein Textfeld, übertragen werden, so hoffen Sie vergebens – es sei denn, Sie hätten daran gedacht, dass für Labels kein `Validating`-Ereignis ausgelöst wird, und haben daher `DataSourceUpdateMode.OnPropertyChanged` eingestellt.

Formatierung

Die Datenbindung funktioniert auch für Eigenschaften, die unterschiedlichen Datentypen angehören. Allerdings muss es dann einen Weg geben, wie die Datentypen ineinander umgewandelt werden können. Nehmen wir z. B. an, Sie möchten die Abmessungen des Fensters (`Size`-Eigenschaft) in ein `TextBox`-Feld einblenden. Die zugehörige Datenbindung könnte wie folgt aussehen:

```
textbox1.DataBindings.Add("Text", this, "Size");
```

Wenn jetzt der Anwender das Fenster vergrößert oder verkleinert, werden die neuen Abmessungen direkt im Textfeld angezeigt. Dies funktioniert, weil die `Size`-Daten automatisch mittels `ToString()` in einen darstellbaren String der Form `"350; 200"` konvertiert werden. Weniger befriedigend sieht die umgekehrte Richtung aus. Entsprechend der Standardeinstellung `DataSourceUpdateMode.OnValidation` sollten Änderungen an der `Text`-Eigenschaft des Textfelds automatisch in die `Size`-Eigenschaft des Fensters übertragen werden, wenn das Textfeld den Fokus verliert. Doch die Übertragung scheitert, und der Grund dafür ist, dass der `Size`-Eigenschaft keine `String`-Daten zugewiesen werden können. Wir müssen also einen Weg vorsehen, wie die `String`-Angaben im Textfeld in ein `Size`-Objekt umgewandelt werden können.

Was nach hohem Aufwand klingt, lässt sich in der Praxis in vielen Fällen recht schnell und unkompliziert erledigen. Meist genügt es, der `Add()`-Methode als viertes Argument `true` zu übergeben, um anzuzeigen, dass der Bindungsmechanismus selbstständig nach geeigneten Konvertierern suchen soll.

```
textbox1.DataBindings.Add("Text", this, "Size", true); // so funktioniert die
// Synchronisierung in beide
// Richtungen
```

Wer mehr Einfluss auf die Konvertierung ausüben möchte oder das Pech hat, dass für die gewünschte Konvertierung kein Standardkonvertierer verfügbar ist (etwa, wenn ein boolescher Wert in eine Farbe umgewandelt werden soll), der kann `Add()` als achtes Argument einen selbst definierten Formatierer übergeben (siehe Kapitel 22) oder eigene Behandlungsmethoden für die Binding-Ereignisse `Format` (Formatierung

der Quelldaten für die Anzeige in der Zieleigenschaft) und Parse (Formatierung der Steuerelementdaten für das Zurückschreiben in die Datenquelle).

Der folgende Code erzeugt die gleiche Datenbindung wie oben, nur dass für die Umwandlung der Text-Daten in ein Size-Objekt eine eigene Parse-Behandlungsmethode definiert wurde:

```
Binding b = new Binding("Text", this, "Size", false);
b.Parse += new ConvertEventHandler(SizeStringToSize);
textBox1.DataBindings.Add(b);
...
private void SizeStringToSize(object sender, ConvertEventArgs e)
{
    if (e.DesiredType != typeof(Size))
        return;

    e.Value = (Size) new SizeConverter().ConvertFromString(e.Value.ToString());
}
```

Hier wird für die Umwandlung einer der vielen von `TypeConverter` abgeleiteten, vordefinierten Konvertierer verwendet. In diesem Fall hätte man den Konvertierer sogar anhand des Typs automatisch bestimmen lassen können:

```
private void SizeStringToSize(object sender, ConvertEventArgs e)
{
    if (e.DesiredType != typeof(Size))
        return;

    e.Value = (Size) TypeDescriptor.GetConverter(typeof(Size))
        .ConvertFromString(e.Value.ToString());
}
```

So dürfte wohl auch der Code aussehen, der erzeugt wird, wenn Sie als viertes Argument `true` übergeben.

Aber auch wenn kein Konvertierer zur Verfügung steht, muss die Implementierung von Parse- oder Format-Behandlungsmethoden nicht schwierig sein, wie das folgende Beispiel demonstriert, welches per Datenbindung den `int`-Wert aus einer vertikalen Bildlaufleiste (Eigenschaft `Value`) ausliest, in einen *Prozentwert* (hier ein `double`-Wert zwischen 0 und 1) konvertiert und dann damit die Transparenz des Formulars (`Opacity`-Eigenschaft) setzt.

```
Binding b = new Binding("Opacity", vScrollBar1, "Value");
b.Format += new ConvertEventHandler(IntToPercent);
this.DataBindings.Add(b);
...
private void IntToPercent(object sender, ConvertEventArgs e)
{
    if (e.DesiredType != typeof(Double))
        return;

    double d = (Int32) e.Value / 100.0;
    e.Value = d;
}
```

Bindungen im Eigenschaftfenster

Grundsätzlich können Sie Datenbindungen auch über das Eigenschaftfenster einrichten. Expandieren Sie dazu im Eigenschaftfenster den (*DataBindings*)-Knoten und klicken Sie anschließend im *Erweitert*-Feld auf die Schaltfläche, um das Dialogfeld *Formatierung und erweiterte Bindung* zu öffnen. Das Dialogfeld ist allerdings vornehmlich für die Einrichtung von Datenbindungen zu Datenbanken oder Objekten konzipiert, für Datenbindungen zwischen Steuerelementen ist es nicht geeignet. Immerhin können Sie aber Datenbindungen zum Formular einrichten.

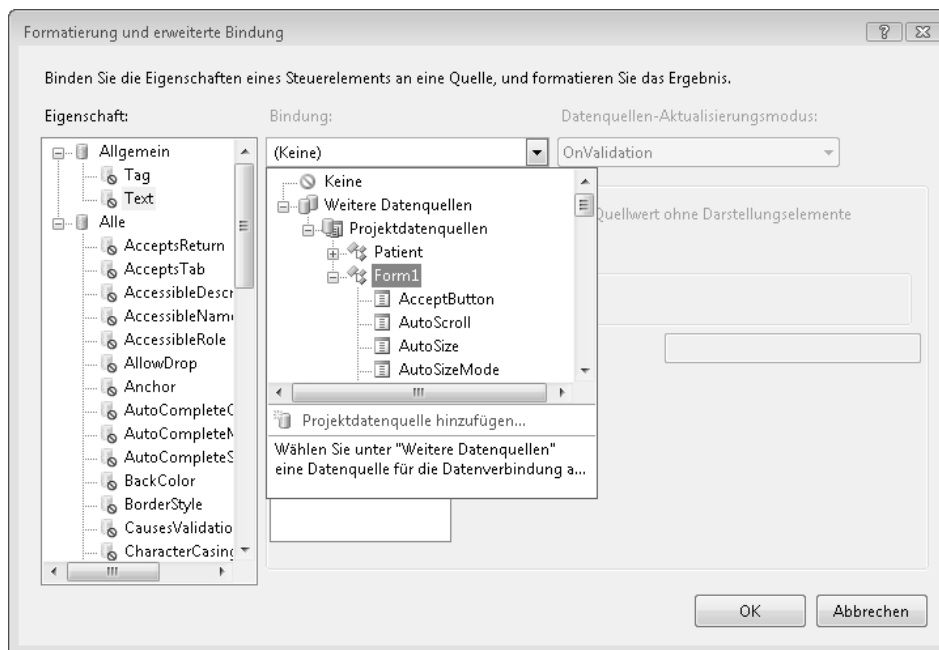


Abbildung 28.7 Das Dialogfeld zur Bearbeitung von Datenbindungen

1. Klicken Sie im Windows Forms-Designer auf das Steuerelement, für das Sie eine Datenbindung einrichten sollen, wechseln Sie in das Eigenschaftfenster und öffnen Sie das Dialogfeld *Formatierung und erweiterte Bindung*.
2. Öffnen Sie das *Bindung*-Listenfeld und klicken Sie auf den Link *Projektdatenquelle hinzufügen*. Wählen Sie *Objekt* aus und im nächsten Dialogfeld den Knoten für das Formular.
3. Anschließend können Sie im *Bindung*-Feld die Formulareigenschaft auswählen, die als Datenquelle dienen soll.
4. Nach Bedarf können Sie den *Datenquellen-Aktualisierungsmodus* und den *Formattyp* einstellen.
5. Die Zieleigenschaft des Steuerelements wählen Sie in der Liste *Eigenschaft* aus.
6. Schließen Sie das Dialogfeld mit *OK*.

Der Windows Forms-Designer richtet daraufhin ein *BindingSource*-Objekt ein, das Datenquellen vom Typ Ihres Formulars verwaltet. Das aktuelle Formular wurde allerdings noch nicht als Datenquelle eingetragen. Dazu müssen Sie:

7. Im Designer-Fenster (Bereich für die nicht sichtbaren Komponenten) nachsehen, wie das Objekt heißt.
8. Im Konstruktor des Formulars dem BindingSource-Objekt mithilfe der Add()-Methode einen Verweis auf das aktuelle Formular-Objekt übergeben:

```
form1BindingSource.Add(this);
```

Einfache Datenbindung zu Objekten, Arrays und Auflistungen

Weiter vorne wurde ausgeführt, dass bei der einfachen Datenbindung nur einfache Werte übertragen werden. Diese können aber natürlich durchaus von »komplexen« Datenquellen stammen. Solche »komplexen« Datenquellen wären beliebige Objekte mit Eigenschaften, Enumerationen⁴ oder jede Datenquelle, die IList implementiert (Array, ArrayList, DataSet etc.).

Datenbindung an Objekte

Die Datenbindung an eine Objekteigenschaft unterscheidet sich nicht wesentlich von der Datenbindung an eine Steuerelement-Eigenschaft.

```
class Customer
{
    private string firstname;
    private string lastname;
    private string id;

    public Customer(string firstname, string lastname, string id)
    {
        Firstname = firstname;
        Lastname = lastname;
        ID = id;
    }

    public string ID
    {
        get { return id; }
        set { id = value; }
    }
    ...
}
...

jim = new Customer("Jim", "Meininger", "HLQ_305");
textBox.DataBindings.Add("Text", jim, "ID");
```

Die hier gezeigte Datenbindung funktioniert allerdings nur in Richtung vom Textfeld zum Objekt, d.h. Änderungen am Inhalt des Textfelds werden in die Objekteigenschaft geschrieben. Zuweisungen an die Objekteigenschaft werden allerdings nicht zum Textfeld geschickt – trotz der Voreinstellung von `Control.UpdateMode.OnPropertyChanged`. Der Grund dafür ist natürlich, dass es zu der Objekteigenschaft ID kein

⁴ Nur bei Vermittlung einer BindingSource.

IDChanged-Ereignis gibt. Wenn Sie das Ereignis als Feld der Klasse Customer definieren und im set-Teil der ID-Eigenschaft die registrierten Ereignisbehandlungsmethoden aufrufen, funktioniert die Datenbindung in beide Richtungen. Aber auch ohne Eingriff in die Klassendefinition ist die Synchronisierung der Datenbindung möglich, siehe Kasten »Manuelle Datensynchronisierung«.

Datenbindung an Arrays

Die einfache Datenbindung zwischen Steuerelementen und einzelnen Array-Elementen folgt dem üblichen Schema. Beachten Sie aber, dass Sie von der Datenbindung nur dann wirklich profitieren können, wenn es sich bei den Array-Elementen um Objekte mit Eigenschaften handelt (vorzugs-, aber nicht notwendigerweise samt Changed-Ereignis).

```
cast = new Figure[3]; // Figure sei eine Klasse mit zwei
cast[0] = new Figure("Detektiv", "Peter Falk"); // Eigenschaften: Figurename und
cast[1] = new Figure("Mörder", "Anthony Perkins"); // Playersname
cast[2] = new Figure("Opfer", "Drew Barrymoore");

tb_4_1.DataBindings.Add("Text", cast[2], "Figurename");
```

Weitaus interessanter ist allerdings folgendes, typisches Szenario: Zu einem Array von Objekten soll eine Benutzeroberfläche geschaffen werden, die es dem Anwender erlaubt, durch das Array der Objekte zu navigieren und dabei die Eigenschaften des aktuell ausgewählten Objekts einzusehen und gegebenenfalls auch zu bearbeiten.

Was aber ist das *aktuelle Objekt* im Array? Arrays besitzen keinen Positionszeiger, keine Methoden zum Vorrücken oder Zurückgehen zum nächsten oder vorangehenden Objekt im Array. Zur Umsetzung der gestellten Aufgabe bedarf es also zuerst einmal einer übergeordneten Kontrollstruktur, die einen Positionszeiger in dem Array verwaltet und am besten auch noch dafür sorgt, dass in den gebundenen Steuerelementen immer die Daten des aktuellen Objekts angezeigt werden. Genau diese Aufgabe übernehmen die Objekte der Windows Forms-Klasse `BindingSource`. Alles, was wir tun müssen, ist, ein Objekt der Klasse zu instanziiieren und zwischen Datenquelle und Steuerelemente zu schalten:

```
// Datenquelle
cast = new Figure[3];
cast[0] = new Figure("Detektiv", "Peter Falk");
cast[1] = new Figure("Mörder", "Anthony Perkins");
cast[2] = new Figure("Opfer", "Drew Barrymoore");

// BindingSource instanziiieren
bs = new BindingSource();

// BindingSource mit Datenquelle verbinden
foreach (Figure f in cast)
    bs.Add(f);

// Steuerelemente an BindingSource binden
tb_5_1.DataBindings.Add("Text", bs, "Figurename");
tb_5_2.DataBindings.Add("Text", bs, "Playersname");
```


HINWEIS Statt die Array-Elemente der `BindingSource` einzeln hinzuzufügen, können Sie auch das Array-Objekt an die `BindingSource`-Eigenschaft `DataSource` zuweisen:

```
bs.DataSource = cast;
```

Zur Navigation im Array und zum Wechseln des aktuellen Objekts stehen Ihnen die `BindingSource`-Methoden `MoveFirst()`, `MovePrevious()`, `MoveNext()` und `MoveLast()` sowie die Eigenschaften `Current` und `Position` zur Verfügung. Ereignisbehandlungsmethoden für passende Navigationsschaltflächen können daher mit nur einer Zeile Code auskommen:

```
private void button1_Click(object sender, EventArgs e)
{
    bs.MoveNext();
}

private void button2_Click(object sender, EventArgs e)
{
    bs.MovePrevious();
}
```

Wenn Sie mit Visual Studio arbeiten, können Sie die `BindingSource` auch als Komponente (Kategorie *Daten*) aus der Toolbox in das Formular ziehen. Sie sparen dann Definition und Instanziierung von `BindingSource` und müssen nur noch die Datenquelle zuweisen und die erzeugte Instanz als Datenquelle bei der Bindung der Steuerelemente angeben.

Und wo wir gerade beim Thema Arbeitersparnis sind: Die Komponente `BindingNavigator` erzeugt eine komplett ausgestattete Symbolleiste mit Navigationsschaltflächen zum Durchlaufen der Elemente in der Datenquelle. Sie müssen lediglich der `BindingSource`-Eigenschaft der Komponente die `BindingSource`-Instanz zuweisen.

Datenbindung an Auflistungen

Die Datenbindung an Auflistungen folgt den gleichen Regeln wie die Datenbindung an Arrays.

Manuelle Datensynchronisierung

Neben der sehr bequemen, automatischen Synchronisierung gibt es auch die Möglichkeit, Datenquelle und gebundenes Steuerelement manuell zu synchronisieren. Mit der `Binding`-Methode `ReadValue()` können Sie den aktuellen Wert der Datenquelle in das Steuerelement schreiben; mithilfe von `WriteValue()` aktualisieren Sie umgekehrt die Datenquelle mit dem Wert aus dem Steuerelement. Um die Methoden aufrufen zu können, benötigen Sie allerdings Zugriff auf das `Binding`-Objekt. Falls Sie das Objekt nicht explizit erzeugt und den Verweis in einem Feld gespeichert haben, können Sie auf das Objekt über die `DataBindings`-Eigenschaft des Steuerelements zugreifen. `DataBindings` ist eine Auflistung, in der die Datenbindungen für alle Eigenschaften des Steuerelements gespeichert sind. Um eine einzelne Bindung herauszugreifen, übergeben Sie den Text der zugehörigen Eigenschaft als Index:

```
textBox.DataBindings["Text"].ReadValue();
```

Eine weitere Möglichkeit, auf die wir allerdings erst im Zusammenhang mit der komplexen Datenbindung detaillierter eingehen, besteht darin, sich über die Form-Eigenschaft `BindingContext` und den Datenquellen-namen als Index eine `BindingManagerBase` zu beschaffen und für diese dann durch Aufruf von `SuspendBinding()` die Datenbindung zunächst auszusetzen, um sie direkt danach mit `ResumeBinding()` wieder anzustoßen.

Komplexe Datenbindung für `ListBox` und `ComboBox`

`ListBox` und `ComboBox` sind sich (nicht nur) hinsichtlich der Datenbindung sehr ähnlich, weswegen hier explizit nur die `ListBox` behandelt wird. Die Ausführungen gelten aber auch für `ComboBox`.

`ListBox` verwaltet eine Gruppe von Objekten und repräsentiert diese durch Strings in seiner Benutzeroberfläche. Wenn der Anwender einen String im Listenfeld auswählt, gibt dieses über verschiedene Eigenschaften den Verweis (`SelectedItem`), den Index (`SelectedIndex`) oder den Wert (`SelectedValue`) des zugehörigen Objekts zurück.

`ListBox`-Steuerelemente können Sie grundsätzlich auf zwei verschiedene Weisen füllen. Die erste Variante sieht so aus, dass Sie die zu verwaltenden Objekte explizit der internen `Items`-Auflistung des Listenfelds hinzufügen. Die Objekte sollten dabei möglichst über eine `String`-Eigenschaft verfügen, die geeignet ist, die einzelnen Objekte zu präsentieren. Den Namen dieser Eigenschaft weisen Sie dann der `ListBox`-Eigenschaft `DisplayMember` zu⁵.

```
List<Figure> cast = new List<Figure>(3);           // Figure sei eine Klasse mit zwei
cast[0] = new Figure("Detektiv", "Peter Falk");    // String-Eigenschaften: Figurename
cast[1] = new Figure("Mörder", "Anthony Perkins"); // und Playersname
cast[2] = new Figure("Opfer", "Drew Barrymoore");

listBox1.Items.Add(cast[0]);
listBox1.Items.Add(cast[1]);
listBox1.Items.Add(cast[2]);
listBox1.DisplayMember = "Figurename";
```

Die Datenbindung herstellen

Die Alternative ist die komplexe Datenbindung an eine passende Datenquelle. Windows Forms-Steuerelemente, die wie `ListBox` die komplexe Datenbindung unterstützen⁶, definieren zu diesem Zweck eine Eigenschaft namens `DataSource`, die einen Verweis auf die zu verwendende Datenquelle erwartet. Doch nicht jede Datenquelle ist für die Zuweisung an `DataSource` geeignet. Passende Datenquellen implementieren die Schnittstelle `IList` (wie z.B. `Array` oder `List<T>`) oder besser noch die abgeleitete Schnittstelle `IBindingList` (wie `DataGridView`) und enthalten eine Sammlung von Objekten. Enumerationen können nur durch Vermittlung einer `BindingSource` gebunden werden. Die komplexe Datenbindung zwischen unserem Listenfeld `listBox1` und der Auflistung `cast` sähe demnach wie folgt aus:

⁵ Wenn es keine passende Objekteigenschaft gibt, die Sie `DisplayMember` zuweisen können, ruft `ListBox` für die einzelnen Objekte `ToString()` auf, um die Anzeigestrings zu generieren. Für Objekte, in deren Klasse `ToString()` nicht überschrieben wurde, bedeutet dies, dass die wenig hilfreiche Typbezeichnung angezeigt wird.

⁶ Das heißt, das Steuerelement zeigt mehrere Daten aus der Datenquelle an.

```
List<Figure> cast = new List<Figure>(3);           // Figure sei eine Klasse mit zwei
cast[0] = new Figure("Detektiv", "Peter Falk");   // String-Eigenschaften: Figurename
cast[1] = new Figure("Mörder", "Anthony Perkins"); // und Playersname
cast[2] = new Figure("Opfer", "Drew Barrymoore");

listBox1.DataSource = cast;
listBox1.DisplayMember = "Figurename";
```

Synchronisierung

Wenn die Datenquelle – wie z. B. die oben verwendete List-Auflistung – nicht die Schnittstelle `IBindingList` implementiert, wird das Listenfeld bei Änderungen an der Datenquelle nicht automatisch aktualisiert. Wenn Sie also im obigen Beispiel nachträglich den Wert von `cast[0].Figurename` ändern, erscheint der neue Name nicht automatisch im Listenfeld. Und wenn Sie gar `cast[0]` eine ganz neue `Figure`-Instanz zuweisen, haben Sie neben der veralteten Anzeige das Problem, dass die Items-Auflistung des Listenfelds immer noch den Verweis auf das Objekt enthält, dessen Verweis ursprünglich in `cast[0]` gespeichert war.

Um Datenquelle und Steuerelement synchron zu halten, müssen Sie daher die Datenbindung vor jeder Änderung an der Datenquelle anhalten und anschließend wieder aufnehmen. Dazu müssen Sie allerdings wissen, dass Windows Forms für jede gebundene Datenquelle ein `BindingManagerBase`-Objekt anlegt. `BindingManagerBase` ist eine abstrakte Klasse mit zwei abgeleiteten Klassen:

- `PropertyManager` für Datenquellen, die aus einem einfachen Eigenschaftswert bestehen
- `CurrencyManager` für Datenquellen, die aus Objektsammlungen bestehen

Die `BindingManagerBase`-Objekte werden in den `BindingContext`-Eigenschaften der Formulare, Container oder Steuerelemente verwaltet und können u. a. zum Aussetzen und Wiederaufnehmen der Datenbindung benutzt werden.

Der folgende Code zeigt, wie Sie die Datenquelle `cast` mit dem Listenfeld `listBox1` synchron halten:

```
listBox_7_1.BindingContext[cast3].SuspendBinding();

cast3[0] = new Figure("Kommissar", "Lino Ventura");

listBox_7_1.BindingContext[cast3].ResumeBinding();()
```

HINWEIS `ListBox`-Listenfelder zeigen nicht nur automatisch alle Objekte aus der gebundenen Datenquelle an, sondern verwalten außerdem auch den Positionszeiger der Datenquelle (vgl. den Einsatz von `BindingSource` für Arrays). Wenn Sie also in einem datengebundenen Listenfeld ein Element auswählen, wird der Positionszeiger automatisch auf das zugehörige Element in der Datenquelle gesetzt. Sollten Sie folglich noch weitere Steuerelemente an die Datenquelle gebunden haben (etwa ein `TextBox`-Steuerelement, das den Wert einer einzelnen Eigenschaft der Objekte aus der Datenquelle anzeigt), beziehen sich diese automatisch auf das Objekt aus der Datenquelle, das in dem Listenfeld ausgewählt ist.

Die Erklärung für dieses durchaus wünschenswerte Verhalten ist, dass die `ListBox` den Positionszeiger nicht selbst setzt, sondern in Wirklichkeit den intern für die Datenquelle angelegten `CurrencyManager` auffordert, dies zu tun – mit der Konsequenz, dass alle Steuerelemente, die über diesen `CurrencyManager` auf die Datenquelle zugreifen, dasselbe aktuelle Objekt verwenden.

Sollte Ihnen dieses Verhalten nicht zusagen, müssen Sie explizit mehrere eigene `CurrencyManager` für Ihre Datenquelle erzeugen. Wenn Sie hingegen feststellen, dass Ihre Steuerelemente nicht in der beschriebenen Art und Weise automatisch synchronisiert werden, sollten Sie prüfen, ob die Datenquelle für die Bindungen immer in der gleichen Weise spezifiziert wurde. Abweichungen können nämlich dazu führen, dass unabsichtlich intern mehrere `CurrencyManager` für die Datenquelle erzeugt werden.

Die `SelectedValue`-Eigenschaft

Wie verwaltet man in einer `ListBox` Objekte, die keine `String`-Eigenschaft für die Darstellung im Listenfeld besitzen? Man erzeugt eine Wrapperklasse mit zwei Eigenschaften: einer `String`-Eigenschaft für die Anzeigetexte und einer Eigenschaft vom Typ der ursprünglichen Objekte. Die Objekte einer solchen Wrapperklasse – oder allgemein jeder Klasse mit zwei Eigenschaften für Anzeigestring und eigentlichen *Wert* – sind die idealen Elemente für eine `ListBox` und der Grund dafür, dass die Klasse `ListBox` neben der Eigenschaft `DisplayMember` auch eine Eigenschaft `SelectedValue` besitzt.

Die Eigenschaft `SelectedValue` kann allerdings nur verwendet werden, wenn die Datenquelle an die `ListBox`-Eigenschaft `DataSource` gebunden wurde und den `ListBox`-Eigenschaften `DisplayMember` und `ValueMember` die Namen der Objekteigenschaften für Anzeigestring und eigentlichen *Wert* übergeben wurde.

```
List<Figure> cast = new List<Figure>(3);           // Figure sei eine Klasse mit zwei
cast[0] = new Figure("Detektiv", "Peter Falk");   // String-Eigenschaften: Figurename
cast[1] = new Figure("Mörder", "Anthony Perkins"); // und Playersname
cast[2] = new Figure("Opfer", "Drew Barrymore");

listBox1.DataSource = cast;
listBox1.DisplayMember = "Figurename";
listBox1.ValueMember = "Playersname";
```

Wenn jetzt ein Element im Listenfeld `listBox1` ausgewählt wird, kann von der Eigenschaft `SelectedValue` der Wert der `Playersname`-Eigenschaft des ausgewählten Objekts abgefragt werden.

Einfache Datenbindung für Listenfelder

`ListBox`-Steuerelemente können aber nicht nur an Datenquellen gebunden werden. Einige ihrer Eigenschaften wie `Text`, `SelectedIndex` oder `SelectedValue` können umgekehrt selbst wieder als Datenquellen für einfache Datenbindungen dienen:

```
// gegeben sei ein ListBox-Steuerelement listBox1
label1.DataBindings.Add("Text", listBox1, "SelectedIndex");
label2.DataBindings.Add("Text", listBox1, "SelectedValue");
```

Die obigen Datenbindungen blenden den Index und den *Wert* des aktuell im Listenfeld ausgewählten Elements in Label-Felder ein.

Verwaltung

Bleiben noch einige Eigenschaften zu erwähnen, die für den Zugriff und die Verwaltung auf die Steuerelemente wichtig sind

Name und Modifiers

Wenn Sie im Windows Forms-Designer ein Steuerelement aus der Toolbox in Ihr Formular ziehen, erzeugt der Designer automatisch eine Instanz des Steuerelements und speichert diese in einem privaten Feld des Formulars.

Den Namen des Felds können Sie im Eigenschaftenfenster über die Eigenschaft (Name), den Zugriffsmodifizierer über die Eigenschaft Modifiers ändern.

Parent und Controls[]

Steuerelemente sind – sieht man einmal von dem Spezialfall des Formulars ab, das ja technisch gesehen ebenfalls ein Steuerelement ist – immer in Container-Steuerelemente oder Formulare eingebettet. Das eingebettete Steuerelement wird relativ zu seinem übergeordneten Container ausgerichtet und auf diesem gezeichnet.

Die Beziehung zwischen Steuerelement und übergeordnetem Container wird in den Eigenschaften Parent (Seite des eingebetteten Steuerelements) und Controls (Seite des übergeordneten Containers) verwaltet. Eine dieser Eigenschaften müssen Sie setzen, wenn Sie Steuerelemente von Hand (ohne Windows Forms-Designer) erzeugen und anzeigen möchten – andernfalls bleibt das Steuerelement unsichtbar:


```
// Code zur manuellen Einbettung eines Label-Steuerelements in ein Formular (this)
Label myLabel = new Label();           // Instanziierung
myLabel.Parent = this;                 // Alternative zu this.Controls.Add(myLabel)
myLabel.Location = new Point(50, 100);
```

Ist die Beziehung zwischen Steuerelement und Container erst einmal hergestellt, können Sie die Eigenschaft Parent verwenden, um vom Steuerelement aus einen Verweis auf den übergeordneten Container zu erhalten. Entsprechend können Sie über die Eigenschaft Controls[] auf jedes eingebettete Steuerelement zugreifen; selbst dann, wenn für das Steuerelement kein Feld in der Formular-Klasse definiert wurde.

Der folgende Code zeigt, wie Sie mithilfe der Controls[]-Eigenschaft auf alle in ein Formular (repräsentiert durch this) eingebetteten Schaltflächen zugreifen können:

```
foreach (Control c in this.Controls)
{
    if (c is Button)
    {
        Button b = c as Button;
        ...
    }
}
```

Statische Anzeigefelder

Beschriftungen – nach dem englischen Sprachgebrauch auch *Labels* genannt – sind Steuerelemente, die einen Text anzeigen. Beschriftungen können vom Anwender nicht editiert werden, übernehmen nicht den Fokus und können konsequenterweise auch nicht mit der -Taste angesteuert werden.

Wie ihr Name andeutet, werden sie vorzugsweise als Beschriftungen für Steuerelemente eingesetzt, die selbst keine eigenen Beschriftungselemente oder Titel enthalten, wie z.B. Eingabefelder oder Listenfelder. Daneben können Beschriftungen aber natürlich auch zum Anzeigen beliebiger Texte verwendet werden – beispielsweise für Erklärungen, Hinweise oder Informationen.

Label-Steuerelemente können auch Bilder anzeigen, doch sind diese meist nur Beiwerk zum Text. Möchten Sie ein Logo oder sonst ein Bild oder eine Grafik in einem Formular anzeigen, ist das PictureBox-Steuerelement die bessere Wahl.

Label

Label-Steuerelemente sind Instanzen der Klasse `Label` aus dem Namespace `System.Windows.Forms`. Sie dienen dazu, einen Text, manchmal auch ein Bild oder eine Kombination aus Bild und Text anzuzeigen.

Der anzuzeigende Text wird der Eigenschaft `Text` zugewiesen:

```
Label myLabel = new Label(); // Instanziierung
myLabel.Parent = this;       // Alternative zu this.Controls.Add(myLabel)
myLabel.Location = new Point(50, 100);

myLabel.Text = "Dies ist ein erläuternder Text";
```

Wenn Sie die Eigenschaft `AutoSize` auf `true` setzen (der Windows Forms-Designer tut dies automatisch), richtet sich die Größe des Label-Steuerelements automatisch nach der Größe des anzuzeigenden Textes. Die Ausrichtung des Textes innerhalb des Label-Steuerelements können Sie über die Eigenschaft `TextAlign` steuern, die Werte der Enumeration `System.Drawing.ContentAlignment` akzeptiert: `BottomCenter`, `BottomLeft`, `BottomRight`, `MiddleCenter`, `MiddleLeft`, `MiddleRight`, `TopCenter`, `TopLeft`, `TopRight`. Der Standardwert ist `TopLeft`.

```
myLabel.TextAlign = System.Drawing.ContentAlignment.TopCenter;
```

Bilder in Label-Steuerelementen

Wenn Sie in einem Label-Steuerelement neben dem Text auch ein Bild, beispielsweise ein kleines Symbol oder Ähnliches, anzeigen wollen, weisen Sie das zugehörige `Image`-Objekt der Eigenschaft `Image` zu und richten Sie das Bild über die Eigenschaft `ImageAlign` aus:

```
myLabel.Image = new Bitmap("Picture01.jpg");
myLabel.ImageAlign = System.Drawing.ContentAlignment.TopLeft;
```

ACHTUNG Die `AutoSize`-Funktion bemisst die optimale Größe eines Label-Steuerelements allein nach dem Text des Steuerelements, nicht nach einem etwaigen Bild. In der Regel empfiehlt sich für Label-Steuerelemente mit eingebundenen Bildern daher die manuelle Dimensionierung.

Mehrzeilige Beschriftungen

Zum Anzeigen mehrzeiliger Texte in einem Label-Steuerelement setzen Sie die Eigenschaft `AutoSize` auf `false`. Der Umbruch erfolgt automatisch am Rand des Steuerelements. Wenn Sie den anzuzeigenden Text im Code an die `Text`-Eigenschaft zuweisen, können Sie Umbrüche mit der Escapesequenz `\n` erzwingen:

```
myLabel.Text = "Erklärender Text, der\n sich über zwei Zeilen erstreckt.";
```

TIPP

Oder verwenden Sie statt eines Beschriftungssteuerelements ein `TextBox`- oder `RichTextBox`-Steuerelement, dessen `Multiline`- und `ReadOnly`-Eigenschaften Sie auf `true` setzen.

LinkLabel

`LinkLabel` ist eine von `Label` abgeleitete Klasse, eine Erweiterung, welche die Anzeige klickbarer Hyperlinks ermöglicht.

Per Voreinstellung ist der gesamte Text eines `LinkLabel`-Steuerelements anklickbar und wird unterstrichen dargestellt. Soll nur ein bestimmter Teil des Texts als Hyperlink fungieren, müssen Sie ein `LinkArea`-Objekt erzeugen, welches den Index des ersten Zeichens und die Länge des Hyperlinks definiert, und dieses an die Eigenschaft `LinkArea` zuweisen.

```
LinkLabel myLinkLabel = new Label(); // Instanziierung
myLinkLabel.Parent = this;           // Alternative zu this.Controls.Add(myLinkLabel)
myLinkLabel.Location = new Point(50, 100);

myLinkLabel.LinkArea = new System.Windows.Forms.LinkArea(28, 46);
```

Klickt der Anwender auf einen `LinkLabel`-Hyperlink wird das `LinkClicked`-Ereignis ausgelöst. Sie können das Ereignis behandeln und mit `System.Diagnostics.Process.Start()` die gewünschte Webseite in den Standardbrowser laden.

```
myLinkLabel.LinkClicked +=
    new LinkLabelLinkClickedEventHandler(myLinkLabel_LinkClicked);

private void myLinkLabel_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    // Website aufrufen
    System.Diagnostics.Process.Start("www.thecompany.com");

    // Hyperlink als "besucht" markieren
    linkLabel1.LinkVisited = true;
}
```

Die Formatierung des Hyperlinks können Sie über die Eigenschaften `ActiveLinkColor`, `DisabledLinkColor`, `LinkColor`, `VisitedLinkColor` und `LinkBehavior` festlegen. Die ersten vier Eigenschaften definieren die Farben für die verschiedenen Hyperlinkzustände und erwarten als Werte `Color`-Objekte. Die Eigenschaft `LinkBehavior` regelt die Unterstreichung und akzeptiert als Wert eine der Konstanten aus der `LinkBehaviour`-Enumeration: `AlwaysUnderline`, `HoverUnderline`, `NeverUnderline`, `SystemDefault`.

```
myLinkLabel.LinkBehavior = LinkBehavior.AlwaysUnderline;
myLinkLabel.LinkColor = System.Drawing.Color.RoyalBlue;
myLinkLabel.VisitedLinkColor = System.Drawing.Color.FromArgb(((int)(((byte)(128)))),
                                                              ((int)(((byte)(0)))),
                                                              ((int)(((byte)(128)))));
```

PictureBox

Bild-Steuerelemente sind Instanzen der Klasse `PictureBox` aus dem Namespace `System.Windows.Forms`. Sie dienen dazu, ein Bild anzuzeigen.

Das anzuzeigende Bild wird der Eigenschaft `Image` zugewiesen:

```
PictureBox myPictureBox = new PictureBox(); // Instanziierung
myPictureBox.Parent = this; // Alternative zu this.Controls.Add(myPictureBox)
myPictureBox.Location = new Point(50, 100);

myPictureBox.Image = new Bitmap("Picture01.jpg");
```

Wie die Größe des Steuerelements und des geladenen Bildes aufeinander abgestimmt wird, können Sie über die Eigenschaft `SizeMode` festlegen, der Sie einen Wert der Enumeration `PictureBoxSizeMode` zuweisen können:

```
// Größe des Bildes an die Größe des Steuerelements anpassen
myPictureBox.SizeMode = PictureBoxSizeMode.StretchImage;
```

Member	Beschreibung
<code>AutoSize</code>	Die Größe des Steuerelements wird an die Größe des Bildes angepasst. (Die Position der linken oberen Ecke bleibt unverändert.)
<code>CenterImage</code>	Das Bild wird im Steuerelement zentriert.
<code>Normal</code>	Das Bild wird an der linken oberen Ecke des Steuerelements ausgerichtet.
<code>StretchImage</code>	Die Größe des Bildes wird an die Größe des Steuerelements angepasst.
<code>Zoom</code>	Das Bild wird unter Beibehaltung seiner ursprünglichen Proportionen an die Größe des Steuerelements angepasst und zentriert.

Tabelle 28.8 Member der `PictureBoxSizeMode`-Enumeration

Schaltflächen

Schaltflächen sind das klassische Steuerelement par excellence. Schaltflächen besitzen einen Titel, sind fokussierbar und reagieren auf Mausklicks. Man unterscheidet zwischen

- klassischen Schaltflächen, die eine bestimmte Aktion ausführen, wenn sie gedrückt werden,
- Kontrollkästchen, die zwischen zwei Zuständen (markiert, nicht markiert) wechseln, wenn sie gedrückt werden,
- Optionsfeldern, die wie Kontrollkästchen zwischen den Zuständen markiert und nicht markiert wechseln, von denen aber immer nur ein Optionsfeld zur Zeit markiert sein kann

Alle Schaltflächen gehen auf die Basisklasse `ButtonBase` zurück, von der sie unter anderem die folgenden Eigenschaften und Ereignisse erben:

ButtonBase-Member	Beschreibung
<code>AutoSize</code>	Passt die Größe der Schaltfläche an den Titel (Text) und – sofern vorhanden – das Bild (Image) an
<code>Click</code>	Ereignis, das ausgelöst wird, wenn die Schaltfläche gedrückt wird
<code>FlatStyle</code>	Ein Wert der <code>FlatStyle</code> -Enumeration, der bestimmt, ob die Schaltfläche dreidimensional oder flach dargestellt wird. Standard – Die Schaltfläche wird dreidimensional dargestellt. System – Die Darstellung wird vom Betriebssystem festgelegt. Popup – Flache Schaltfläche, die in eine erhabene Darstellung wechselt, wenn die Maus über sie bewegt wird. Flat – Flache Schaltfläche. Die Darstellung kann über die Eigenschaft <code>FlatAppearance</code> angepasst werden.
<code>Image</code> und <code>ImageAlign</code>	Festlegung und Ausrichtung eines in der Schaltfläche anzuzeigenden Bildes. Die Ausrichtung wird durch einen Wert der Enumeration <code>ContentAlignment</code> festgelegt, siehe Abschnitt zu Label.
<code>Text</code> und <code>TextAlign</code>	Festlegung und Ausrichtung des Schaltflächentitels. Die Ausrichtung wird durch einen Wert der Enumeration <code>ContentAlignment</code> festgelegt, siehe Abschnitt zu Label.
<code>TextImageRelation</code>	Eigenschaft, die die Position des Bildes relativ zum Titel festlegt. Akzeptiert die Werte der <code>TextImageRelation</code> -Enumeration: <code>ImageAboveText</code> , <code>ImageBeforeText</code> , <code>Overlay</code> (Standard), <code>TextAboveImage</code> , <code>TextBeforeImage</code> .

Tabelle 28.9 Allgemeine Schaltflächen-Member

Klassische Schaltflächen

Klassische Schaltflächen sind Instanzen der Klasse `Button` aus dem Namespace `System.Windows.Forms`. Ihre Aufgabe ist es, eine bestimmte Aktion auszuführen, wenn sie gedrückt werden. Das Drücken einer Schaltfläche wird automatisch animiert, wenn der Anwender die Schaltfläche anklickt.

Der folgende Code erzeugt die Schaltfläche aus Abbildung 28.8.

```
public Form1()
{
    Button myButton = new Button();    // Instanziierung
    myButton.Parent = this;           // Einbettung in Formular

    // Titeltext und Bild, Letzteres wird aus Ressourcendatei geladen
    myButton.Text = "Klick mich";
    myButton.Image = global::Schaltflächen.Properties.Resources.ButtonImage;
```

```
// Ausrichtung von Bild und Text
myButton.TextImageRelation = TextImageRelation.ImageBeforeText;
myButton.Padding = new Padding(50, 0, 50, 0);
myButton.AutoSize = true;

// Schaltfläche horizontal zentriert anzeigen
myButton.Location = new Point((ClientSize.Width - myButton.Width) / 2,
                               (ClientSize.Height - 50));



// Ereignisbehandlung
myButton.Click += new EventHandler(myButton_Click);
}

private void myButton_Click(object sender, EventArgs e)
{
    // Beim Anklicken der Schaltfläche Beep-Ton ausgeben
    System.Media.SystemSounds.Beep.Play();
}
```



Abbildung 28.8 Zentrierte Schaltfläche mit Bild

Unterstützung der Eingabetaste

Wenn der Anwender die -Taste drückt und der Fokus gerade auf einer Schaltfläche liegt, wird deren Click-Behandlungsmethode ausgeführt. Wenn der Fokus nicht auf einer Schaltfläche liegt (und auch nicht auf einem Steuerelement, das auf die -Taste reagiert), passiert nichts weiter. Es sei denn, Sie hätten eine Schaltfläche des Formulars als Standardschaltfläche festgelegt. In diesem Falle würde die Click-Behandlungsmethode der Standardschaltfläche ausgeführt.

Um eine Schaltfläche zur Standardschaltfläche zu erheben, weisen Sie die Schaltfläche der AcceptButton-Eigenschaft des Formulars zu.

```
this.AcceptButton = myButton;
```

Schaltflächen deaktivieren

Je nach Bearbeitungszustand eines Formulars werden Sie vielleicht einzelne Schaltflächen-Aktionen sperren wollen. So wird z. B. die *Weiter*-Schaltfläche von Lizenz-Dialogfeldern meist erst dann freigegeben, wenn der Anwender auf irgendeine Weise bestätigt, dass er die Lizenzvereinbarungen gelesen und akzeptiert hat.

Um Schaltflächen zur Entwurfs- oder Laufzeit zu deaktivieren, setzen Sie die `Enabled`-Eigenschaft der Schaltfläche auf `false`. Entsprechend weisen Sie der `Enabled`-Eigenschaft `true` zu, wenn die Schaltfläche wieder aktiviert werden soll.

```
myButton.Enabled = false;
```

HINWEIS Deaktivierte Schaltflächen werden automatisch grau dargestellt und interagieren nicht mehr mit dem Anwender.

Kontrollkästchen

Kontrollkästchen sind Instanzen der Klasse `CheckBox` aus dem Namespace `System.Windows.Forms`. Sie repräsentieren Einstellungen, die der Anwender durch Klick ein- oder ausschalten kann.



Abbildung 28.9 Kontrollkästchen

Kontrollkästchen verfügen über eine eigene Beschriftung, die Sie über die Eigenschaft `Text` setzen können. Der Zustand des Kontrollkästchens kann über die Eigenschaft `Checked` gesetzt und abgefragt werden.

```
CheckBox myCheckBox = new CheckBox(); // Instanziierung
myCheckBox.Parent = this;             // Alternative zu this.Controls.Add(myCheckBox)
myCheckBox.Location = new Point(80, 50);

myCheckBox.Text = "Salami";
myCheckBox.Checked = true;
```

Kontrollkästchen mit drei Zuständen

Wenn Sie die `ThreeState`-Eigenschaft des Kontrollkästchens auf `true` setzen, wechselt das Kontrollkästchen beim Anklicken zwischen drei statt zwei Zuständen. (Der dritte Zustand wird als gefülltes Kästchen angezeigt.)

Der aktuelle Zustand eines `ThreeState`-Kontrollkästchens kann über die Eigenschaft `CheckState` abgefragt werden, die einen Wert der Enumeration `CheckState` zurückliefert: `Checked`, `Unchecked`, `Indeterminate`.

```
if (checkBox1.CheckState == CheckState.Indeterminate)
```

Im `Indeterminate`-Zustand hat die Eigenschaft `Checked` den Wert `true`.

Ereignisse

Kontrollkästchen-Ereignisse werden eher selten abgefangen. Ihr Zustand wird meist erst ausgewertet, wenn das übergeordnete Dialogfeld beendet wird. Mit den Ereignissen `Click`, `CheckedChanged` und `CheckStateChanged` können Sie aber auch sofort auf das Anklicken reagieren.

Optionsfelder und GroupBox

Das Gegenstück zu den Kontrollkästchen sind die Optionsfelder. Sie besitzen ein rundes Markierungsfeld und erlauben immer nur die Auswahl eines Optionsfelds.



Abbildung 28.10 Optionsfelder

Optionsfelder verfügen über eine eigene Beschriftung, die Sie über die Eigenschaft `Text` setzen können. Der Zustand des Optionsfelds kann über die Eigenschaft `Checked` gesetzt und abgefragt werden.

```
RadioButton myRadioButton = new RadioButton(); // Instanziierung
myRadioButton.Parent = this;                  // oder this.Controls.Add(myRadioButton)
myRadioButton.Location = new Point(17, 33);

myRadioButton.Text = "Schwarz";
myRadioButton.Checked = true;
```

Gruppierung

Alle Optionsfelder, die zusammen in einen Container eingebettet sind, bilden eine Gruppe, aus der jeweils nur ein einziges Optionsfeld ausgewählt werden kann. Wenn Sie innerhalb eines Containers, etwa in einem Dialogfeld, mehrere Gruppen von Optionsfeldern anbieten wollen, richten Sie für jede Gruppe einen `GroupBox`-Container ein und fügen diesem die Optionsfelder der Gruppe hinzu (siehe Abbildung 28.10).

```
GroupBox groupBox1 = new GroupBox();
groupBox1.Name = "groupBox1";
groupBox1.Parent = this;
groupBox1.Text = "Vordergrundfarbe";
groupBox1.Controls.Add(this.radioButton3);
groupBox1.Controls.Add(this.radioButton2);
groupBox1.Controls.Add(this.radioButton1);
groupBox1.Location = new Point(36, 28);
groupBox1.Size = new Size(138, 108);
```

```
GroupBox groupBox2 = new GroupBox();
groupBox2.Name = "groupBox2";
groupBox2.Text = "Hintergrundfarbe";
groupBox2.Parent = this;
groupBox2.Controls.Add(this.radioButton4);
groupBox2.Controls.Add(this.radioButton5);
groupBox2.Controls.Add(this.radioButton6);
groupBox2.Location = new System.Drawing.Point(194, 28);
groupBox2.Size = new System.Drawing.Size(138, 108);
```

Ereignisse

Optionsfelder-Ereignisse werden eher selten abgefangen. Ihr Zustand wird meist erst ausgewertet, wenn das übergeordnete Dialogfeld beendet wird. Mit den Ereignissen `Click` und `CheckedChanged` können Sie aber auch sofort auf das Anklicken reagieren.

Textfelder

Zur Textverarbeitung, unabhängig davon, ob es sich dabei um das einfache Einlesen eines Namens oder die Bearbeitung eines Textdokuments handelt, gibt es in der Windows Forms-Bibliothek eine Reihe von nützlichen Steuerelementen.

TextBox

Instanzen der Klasse `System.Windows.Forms.TextBox` sind recht leistungsfähig, unterstützen standardmäßig die Zwischenablage und sind vielseitig einsetzbar, beispielsweise als einzeilige Eingabefelder oder als mehrzeilige Variante zum Editieren von Texten.

Eingabefelder

In der Standardkonfiguration sind `TextBox`-Felder einzeilige Eingabefelder. Über die Eigenschaft `Text` können Sie einen Text in dem Textfeld ausgeben oder umgekehrt den vom Anwender eingegebenen Text auslesen.

Wenn der Anwender einen Text eingibt, der länger ist als die Breite des Textfelds, wird die Anzeige automatisch gescrollt. Wenn Sie stattdessen lieber eine maximale Zahl von erlaubten Zeichen vorgeben möchten, weisen Sie diese der Eigenschaft `MaxLength` zu.

Wenn Sie möchten, dass der Text im Textfeld nicht vom Anwender bearbeitet werden kann, setzen Sie die Eigenschaft `ReadOnly` auf `true`.

Wenn Sie verhindern möchten, dass der Anwender den Inhalt der Zwischenablage in das Eingabefeld einfügen oder umgekehrt den Text des Eingabefelds in die Zwischenablage kopieren kann, setzen Sie die Eigenschaft `ShortcutsEnabled` auf `false`.

```
TextBox textBox1 = new TextBox(); // Instanziierung
textBox1.Parent = this;           // Alternative zu this.Controls.Add(textBox1)
textBox1.Location = new Point(50, 100);
```

```
textBox1.Text = "Ihr Name";
textBox1.MaxLength = 10;
```

Um den Inhalt des Eingabefelds von der Anwendung aus zu löschen, weisen Sie Text einfach einen leeren String "" zu oder rufen Sie die Methode `Clear()` auf. Wenn Sie sich vor dem Auslesen des Textes über die Länge in Zeichen informieren wollen, können Sie diese über die Eigenschaft `TextLength` abfragen. Wenn Sie nicht den gesamten Text im Eingabefeld auslesen möchten, sondern nur den Teil, der markiert ist, fragen Sie die Eigenschaft `SelectedText` ab.


Für die Überwachung von Eingaben gibt es zwei sehr interessante Ereignisse. Das `TextChanged`-Ereignis wird jedes Mal ausgelöst, wenn sich der Inhalt im Eingabefeld ändert – Sie können über dieses Ereignis also jedes Zeichen kontrollieren, welches der Anwender gerade eintippt.

Wenn Sie den Inhalt des Eingabefelds nach Abschluss der Eingabe (wenn der Anwender den Fokus weiterreicht) validieren möchten, achten Sie darauf, dass die Eigenschaft `CausesValidation` auf `true` gesetzt ist und fangen Sie das `Validating`-Ereignis ab:

```
textBox1.CausesValidation = true;
textBox1.Validating +=
    new System.ComponentModel.CancelEventHandler(this. textBox1_Validating);

// sicherstellen, dass mindestens fünf Zeichen eingegeben wurden
private void textBox1_Validating(object sender, CancelEventArgs e)
{
    if (textBox1.TextLength < 5)
    {
        textBox1.Text = "";    // Eingabe löschen
        textBox1.Focus();      // Fokus zurücksetzen
        MessageBox.Show("Die Eingabe muss mindestens 5 Zeichen umfassen.");
    }
}
```

Mehrzeilige Textfelder

Soll sich das Textfeld über mehrere Zeilen erstrecken, müssen Sie die Eigenschaft `Multiline` auf `true` setzen und das Textfeld entsprechend höher setzen. Über die Eigenschaft `ScrollBars` können Sie das Textfeld mit Bildlaufleisten ausstatten. Textzeilen, die über das Ende des Textfelds hinausgehen, werden automatisch umgebrochen, es sei denn, Sie setzen `WordWrap` auf `false`. Wenn Sie Tabulatoren akzeptieren und das Drücken der -Taste als Beginn einer neuen Zeile interpretieren möchten, setzen Sie die Eigenschaften `AcceptsTab` und `AcceptsReturn` auf `true`.

```
TextBox textBox1 = new TextBox();
textBox1.Parent = this;
textBox1.Location = new Point(50, 100);

textBox1.Multiline = true;
textBox1.AcceptsReturn = true;
textBox1.AcceptsTab = true;
textBox1.ScrollBars = ScrollBars.Vertical;
```

Mithilfe der Methode `AppendText()` können Sie einen weiteren Text an den bestehenden Text des Textfelds anhängen. Alternativ können Sie den Text im Steuerelement auch mithilfe des String-Arrays `Lines` zeilenweise bearbeiten.

HINWEIS Unter den Beispielanwendungen zu diesem Buch finden Sie auch einen einfachen Texteditor, der eine `TextBox`-Instanz zum Anzeigen und Editieren von Textdateien verwendet.

Passwortheingaben

Werden geheime Daten über ein Eingabefeld abgefragt, gehört es sich, dass Sie die eingegebenen Zeichen kaschieren. Weisen Sie dazu einfach der `PasswordChar`-Eigenschaft das Zeichen zu, welches anstelle der eingegebenen Zeichen angezeigt werden soll.

```
textBox1.PasswordChar = '*';
```

MaskedTextBox

Maskierte Eingabefelder sind Instanzen der Klasse `MaskedTextBox` aus dem Namespace `System.Windows.Forms`. Es sind spezialisierte Eingabefelder, die mittels einer Eingabemaske sicherstellen, dass die Eingabe ein bestimmtes Format hat.

Die Maske weisen Sie der Eigenschaft `Mask` zu. Im Eigenschaftfenster können Sie zwischen einer Reihe von vordefinierten oder der Erstellung einer eigenen Maske wählen. In einer Maskendefinition können Sie neben normalen Zeichen, die für sich selbst stehen, unter anderem folgende Platzhalter verwenden:

Platzhalter	Beschreibung	Platzhalter	Beschreibung
0	Obligatorische Ziffer	9	Optionale Ziffer (oder Leerzeichen)
L	Obligatorischer Buchstabe	?	Optionaler Buchstabe
&	Obligatorisches Zeichen	C	Optionales Zeichen
A	Obligatorisches alphanum. Zeichen	a	Optionales alphanum. Zeichen
.	Trennzeichen für Dezimalstellen	,	Tausenderzeichen
:	Zeittrennzeichen	/	Datumstrennzeichen
\$	Währungssymbol	\	Escape-Zeichen

Tabelle 28.10 Einige Platzhalter für `MaskedTextBox`-Masken

```
00/00/0000 // Datumsmaske
000099      // Postleitzahl mit vier bis sechs Ziffern
LL-LL      // Kürzel für eine Lokale
```

Die Platzhalter aus der Maske werden im Eingabefeld durch einen Unterstrich oder jedes beliebige Zeichen ersetzt, welches Sie der Eigenschaft `PromptChar` zuweisen.

```
MaskedTextBox myMaskedTextBox = new MaskedTextBox();  
myMaskedTextBox.Parent = this;  
  
myMaskedTextBox.Mask = "999-99999";
```



Abbildung 28.11 Dialog mit Masken-Eingabefeld

Das Masken-Eingabefeld stellt selbstständig sicher, dass für einen Platzhalter nur zulässige Zeichen akzeptiert werden. Es überprüft aber beim Verlassen des Eingabefelds nicht, ob die Maske vollständig ausgefüllt wurde. Hierfür müssen Sie die boolesche Eigenschaft `MaskFull` abfragen.

RichTextBox

Instanzen der Klasse `RichTextBox` aus dem Namespace `System.Windows.Forms` werden vornehmlich zur Bearbeitung von Textdateien verwendet. Sie unterstützen die Formatierung des Textes und stellen Methoden zum Laden und Speichern von Dateien zur Verfügung.

Drehfelder, Listenfelder und Kombinationsfelder

Diese Steuerelemente ermöglichen dem Anwender die Auswahl eines Wertes aus einer vorgegebenen Menge von Werten.

Drehfelder

Drehfelder, auch als Auf-Ab-Steuerelemente bezeichnet, sind Instanzen der Klasse `NumericUpDown` aus dem Namespace `System.Windows.Forms`. Sie erlauben die Auswahl von Zahlenwerten, die der Anwender direkt eingeben oder durch Betätigen der Pfeilschalter im Steuerelement bzw. der Pfeiltasten auf der Tastatur auswählen kann.

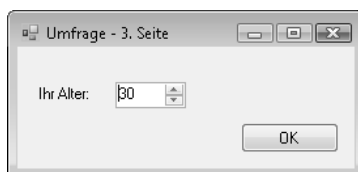


Abbildung 28.12 Drehfelder

Der Wert eines Drehfelds wird durch die Eigenschaft `Value` repräsentiert und kann auf Zahlen zwischen `Minimum` und `Maximum` eingegrenzt werden. Die Position der Pfeiltasten legt die Eigenschaft `UpDownAlign` fest, die einen der Werte `Left` oder `Right` der Enumeration `LeftRightAlignment` annehmen kann (Vorgabe ist `Right`). Wenn Sie verhindern möchten, dass der Wert im Drehfeld durch Betätigen der Pfeiltasten verändert werden kann, setzen Sie die Eigenschaft `InterceptArrowKeys` auf `false`.

```

NumericUpDown myNumericUpDown = new NumericUpDown(); // Instanziierung
myNumericUpDown.Parent = this;                       // oder this.Controls.Add(myNumericUpDown)
myNumericUpDown.Location = new Point(75, 30);

myNumericUpDown.Maximum = 110;                        // Maximalwert
myNumericUpDown.Minimum = 1;                         // Minimalwert
myNumericUpDown.Value = 30;                          // Vorgabe
myNumericUpDown.InterceptArrowKeys = false;

```

Wenn Sie ein Drehfeld zur Auswahl von Gleitkommawerten benötigen, weisen Sie der `DecimalPlaces`-Eigenschaft für die Anzahl der Dezimalstellen einen Wert größer 0 zu.

Wenn Sie ein Drehfeld zur Auswahl von Hexadezimalzahlen benötigen, setzen Sie die Eigenschaft `Hexadecimal` auf `true`.

Listenfelder

Listenfelder sind Instanzen der Klasse `ListBox` aus dem Namespace `System.Windows.Forms`. Sie erlauben dem Anwender aus einer vorgegebenen Liste von Objekten eines oder mehrere Objekte auszuwählen. Die Objekte werden im Listenfeld durch Strings repräsentiert, wobei es in der Verantwortung der einzelnen Objekte liegt, passende Strings zu liefern. Wenn die Objekte über eine `String`-Eigenschaft verfügen, deren Inhalt geeignet ist, weisen Sie den Namen der Eigenschaft der `ListBox`-Eigenschaft `DisplayMember` zu. Ansonsten erzeugt `ListBox` die Strings durch Aufruf der `ToString()`-Methode der Objekte. Ist die `Sorted`-Eigenschaft auf `true` gesetzt, werden die Elemente in sortierter Reihenfolge aufgelistet. Gerät die Reihenfolge durcheinander, kann sie durch Aufruf der `Sort()`-Methode wiederhergestellt werden.

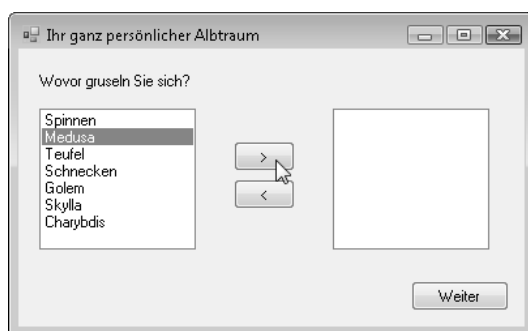


Abbildung 28.13 Listenfelder

Elemente hinzufügen

Listenfelder verwahren ihre Elemente in der Eigenschaft `Items`, die vom Typ `ObjectCollection` ist. Sofern Sie Strings als Elemente einfügen wollen, können Sie in Visual Studio über das Eigenschaftsfenster ein Dialog-

feld aufrufen, in dem Sie nur noch die Zeichenfolgen eintippen müssen. Für Elemente anderer Typen sowie in Fällen, wo das Eigenschaftfenster nicht zur Verfügung steht, können Sie Elemente mit den `ObjectCollection`-Methoden `Add()`, `Insert()` und `AddRange()` hinzufügen. `Add()` und `AddRange()` fügen die Elemente am Ende der `Items`-Auflistung an, `Insert()` übernimmt als erstes Argument einen Index, der zwischen 0 und `Items.Count` liegen muss.

```
ListBox myListBox = new ListBox(); // Instanziierung
myListBox.Parent = this; // Alternative zu this.Controls.Add(myListBox)
myListBox.Location = new Point(50, 100);

myListBox.Items.Add("Spinnen");
myListBox.Items.Add("Medusa");
//...
```

Oder mit `AddRange()`:

```
myListBox.Items.AddRange(new object[] { "Spinnen", "Medusa", "Teufel",
                                         "Schnecken", "Golem", "Skylia", "Charybdis" });
```

Auf einzelne Elemente im Listenfeld können Sie per Index zugreifen oder indem Sie die `Items`-Auflistung mit `foreach` durchlaufen.

HINWEIS Unveränderliche Listen können Sie über die `DataSource`-Eigenschaft mit den Werten aus einem Array oder einem beliebigen Objekt füllen, dessen Typ `IList` oder `IListSource` implementiert. Ausführliche Informationen zur Verwendung der `DataSource`-Eigenschaft finden Sie weiter unten im Abschnitt »Beliebige Objekte als Elemente« und in den Ausführungen zur Datenbindung im einleitenden Abschnitt dieses Kapitels.

```
string[] capitals = { "Rom", "Helsinki", "Oslo", "Paris" };
capitalListBox.DataSource = capitals;

enum Towns { Amsterdam, Berlin, Oslo, Paris };
myListBox.DataSource = Enum.GetValues(typeof(Towns));
```

Ausgewählte Elemente abfragen

Über die Eigenschaft `SelectionMode` können Sie festlegen, wie und wieviele Elemente im Listenfeld gleichzeitig markiert werden können.

Wert	Beschreibung
None	Es können keine Elemente ausgewählt werden.
Single	Es kann immer nur ein Element zurzeit ausgewählt werden.
MultiSingle	Es können mehrere Elemente gleichzeitig ausgewählt werden. Jeder Klick mit der Maustaste wählt ein Element aus oder hebt die Auswahl eines markierten Elements auf.
MultiExtended	Es können mehrere Elemente gleichzeitig ausgewählt werden. Die Auswahl mit <code>Strg</code> - und <code>⇧</code> -Taste wird unterstützt.

Tabelle 28.11 Werte der Enumeration `SelectionMode`

Wie können Sie abfragen, welches Element (welche Elemente) in einem Listenfeld ausgewählt wurde? Handelt es sich um ein Listenfeld mit Einfachauswahl, speichert die `SelectedItem`-Eigenschaft einen Verweis auf das

Element und die `SelectedIndex`-Eigenschaft den Index. Für Listenfelder mit Mehrfachauswahl liefern diese Eigenschaften die Daten eines nicht näher bestimmten Elements aus der Auswahl. Die vollständige Auswahl liefern die Eigenschaften `SelectedItems` und `SelectedIndices` – in Form einer `SelectedObjectCollection`-Auflistung von Objektverweisen oder einer `SelectedIndicesCollection`-Auflistung von ganzzahligen Indices.

Das folgende Codefragment definiert eine Ereignisbehandlungsmethode zu einer Schaltfläche, die die im `listBox1`-Listenfeld ausgewählten Elemente in ein zweites Listenfeld `listBox2` kopiert.

```
private void button1_Click(object sender, EventArgs e)
{
    if (listBox1.SelectedItems.Count > 0)
    {
        foreach (object o in listBox1.SelectedItems)
        {
            listBox2.Items.Add(o);
        }
    }
}
```

Bei einem Listenfeld mit Einfachauswahl würde sich dieser Code vereinfachen zu:

```
if (listBox1.SelectedItem != null)
{
    listBox2.Items.Add( listBox1.SelectedItem );
}
```

Elemente löschen

Mit der `Remove()`-Methode können Sie Elemente aus den internen Auflistungen des Listenfelds löschen. Elemente aus der `Items`-Auflistung können Sie löschen, indem Sie `Remove()` den Verweis auf das Objekt oder `RemoveAt()` den Index des Objekts übergeben. Aus den Auflistungen für die ausgewählten Elemente (`SelectedItems` und `SelectedIndices`) können Sie Einträge nur mit `Remove(obj)` bzw. `Remove(index)` entfernen.

Bei Listenfeldern mit Mehrfachauswahl ist allerdings äußerste Vorsicht geboten, denn die Listen sind nicht unabhängig voneinander. Wenn Sie beispielsweise ein markiertes Listenelement aus der `Items`-Auflistung löschen, wird es sofort auch aus der `SelectedItems`-Auflistung getilgt. Das nachfolgende Codefragment, welches die ausgewählten Elemente von einer Liste in eine zweite Liste verschiebt, berücksichtigt dies. Es verwendet zum Löschen eine zweite Schleife, die `SelectedItems.Count`-Mal das erste Element in der `SelectedItems`-Auflistung löscht.

```
private void button1_Click(object sender, EventArgs e)
{
    if (listBox1.SelectedItems.Count > 0)
    {
        foreach (object o in listBox1.SelectedItems)
        {
            listBox2.Items.Add(o);
        }
    }
}
```

```

        int count = listBox1.SelectedItems.Count;
        for(int i = 0; i < count; i++)
        {
            object o = listBox1.SelectedItems[0];
            listBox1.Items.Remove(o);
        }
    }
}

```

Bei einem Listenfeld mit Einfachauswahl würde sich dieser Code vereinfachen zu:

```

if (listBox1.SelectedItem != null)
{
    object s = listBox1.SelectedItem;
    listBox2.Items.Add(s);
    listBox1.Items.Remove(s);
}

```

Beliebige Objekte als Elemente

Meist werden Strings als Elemente in Listenfeldern abgelegt. Sie können aber auch beliebige Objekte in ein Listenfeld einfügen. Im Listenfeld werden dann die ToString()-Rückgabewerte der Objekte angezeigt. Es sei denn, die Objekte besitzen eine String-Eigenschaft, die Sie als DisplayMember festlegen.

Wenn Sie die Objekte in Form eines Arrays oder einer IList-Auflistung an die DataSource-Eigenschaft des Listenfelds zuweisen, können Sie zudem neben der DisplayMember- auch noch eine ValueMember-Eigenschaft (beliebigen Typs) festlegen. Über SelectedValue können Sie dann genau diese Eigenschaft für das aktuell ausgewählte Element abfragen (oder setzen).

```

class Person
{
    string firstname;
    string secondname;
    int age;

    public Person(string firstname, string secondname, int age)
    {
        this.firstname = firstname;
        this.secondname = secondname;
        this.age = age;
    }

    public string Secondname
    {
        get { return secondname; }
    }

    public int Age
    {
        get { return age; }
    }
}

```

```
// ...

ListBox listBox = new ListBox();
Person[] friends = {new Person("Gloria", "Mint", 32),
                    new Person("Rainer", "Helsk", 45)};

listBox.DataSource = friends;
listBox.DisplayMember = "Secondname";
listBox.ValueMember = "Age";
listBox.SelectedIndex = 1;
MessageBox.Show(listBox.SelectedValue.ToString());
```

Ereignisse

Wenn Sie direkt auf Änderungen in der Auswahl reagieren möchten, behandeln Sie die Ereignisse `SelectedIndexChanged` und `SelectedValueChanged`.

Kombinationsfelder

Kombinationsfelder sind Instanzen der Klasse `ComboBox` aus dem Namespace `System.Windows.Forms`. Ein Kombinationsfeld ist eine Kombination aus einem Eingabefeld und einem Listefeld mit Einfachauswahl. Der Anwender kann wahlweise ein Listenelement markieren, worauf dieses in das Eingabefeld übernommen wird, oder selbst einen Text in das Eingabefeld eingeben.

Über die Eigenschaften `Text` und `SelectedText` können Sie den Inhalt des Eingabefelds abfragen (oder setzen). Die Darstellung des Kombinationsfelds können Sie über die Eigenschaft `DropDownStyle` festlegen, die Werte der Enumeration `ComboBoxStyle` akzeptiert.

Wert	Beschreibung
DropDown (Standard)	Kombination aus Eingabefeld und zusammengeklappter Liste.
DropDownList	Kombination aus schreibgeschütztem Textfeld und zusammengeklappter Liste.
Simple	Kombination aus Eingabefeld und aufgeklappter Liste.

Tabelle 28.12 Werte der Enumeration `ComboBoxStyle`

```
ComboBox myComboBox = new ComboBox(); // Instanziierung
myComboBox.Parent = this; // oder this.Controls.Add(myComboBox)
myComboBox.Location = new Point(50, 100);

myComboBox.DropDownStyle = ComboBoxStyle.Simple;
myComboBox.Items.AddRange(new object[] { "1", "2", "3" });
```

Datumsauswahl

Das Abfragen von Datums- und Zeitangaben ist grundsätzlich eine eher heikle Aufgabe, da die Datums- und Zeitwerte vom Anwender in einem Format angegeben werden müssen, das die Umwandlung in `DateTime`-Objekte zur weiteren Verarbeitung innerhalb der Anwendung erlaubt.

Um das Abfragen und Einlesen von Datums- und Zeitangaben sicherer und nebenbei auch für den Anwender ansprechender zu gestalten, gibt es in der Windows Forms-Bibliothek drei Steuerelemente, mit denen vorformatierte Datumsangaben und Uhrzeiten eingelesen werden. Neben dem `MaskedTextBox`-Steuerelement, das bereits im Abschnitt »Textfelder« vorgestellt wurde, sind dies insbesondere `MonthCalendar` und `DateTimePicker`, die gegenüber `MaskedTextBox` den Vorteil haben, dass sie dem Anwender eine grafische Oberfläche (Kalenderblatt) zur Auswahl des Datums anbieten und das Ergebnis bereits als `DateTime`-Objekt zurückliefern. `DateTimePicker` besitzt im Unterschied zu `MonthCalendar` ein aufklappbares Monatsfeld und erlaubt auch die Auswahl von Uhrzeiten.

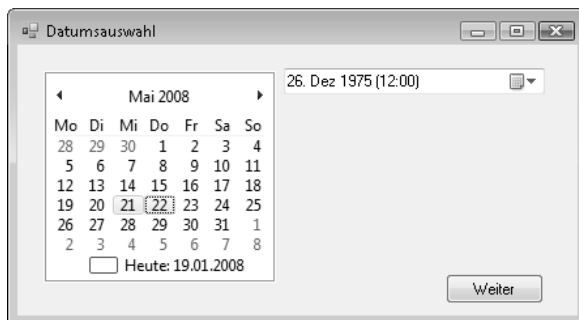


Abbildung 28.14 `MonthCalendar` und `DateTimePicker`

MonthCalendar

Das `MonthCalendar`-Steuerelement präsentiert dem Anwender ein Kalenderblatt, über das er ein einzelnes Datum oder eine Zeitperiode auswählen kann. Welches Datum oder welche Zeitperiode anfangs ausgewählt wird, legen Sie über die `SelectionRange`-Eigenschaft fest. Hinter dieser Eigenschaft verbirgt sich ein `SelectionRange`-Objekt, das zwei `DateTime`-Felder für den ersten (Start) und den letzten (End) ausgewählten Tag definiert. Wenn Sie für Start und End das gleiche Datum angeben, wird nur der eine Tag ausgewählt. Sie müssen die Zuweisung allerdings im Konstruktor nach Aufruf der `InitializeComponent()`-Methode vornehmen, da das Eigenschaftfenster bei gleichen Start- und End-Werten den Standardwert, d.h. das aktuelle Datum, als Vorauswahl verwendet.

```
MonthCalendar myMonthCalendar = new MonthCalendar(); // Instanziierung
myMonthCalendar.Parent = this;                       // oder this.Controls.Add(myMonthCalendar)
myMonthCalendar.Location = new Point(20, 20);

myMonthCalendar.SelectionRange = new SelectionRange(new System.DateTime(2006, 5, 21),
                                                    new System.DateTime(2006, 5, 22));
```

Darstellung und Funktion des Kalenders können durch viele spezielle Eigenschaften angepasst werden:

- Das aktuelle Datum ...
 - ... wird im Kalenderblatt rot eingekreist und zusätzlich noch einmal am unteren Rand eingeblendet. Um die Markierung zu entfernen, setzen Sie `ShowTodayCircle` auf `false`. Um die Einblendung zu entfernen, setzen Sie `ShowToday` auf `false`. Soll statt des Datums des aktuellen Tages ein anderes Datum angezeigt und eingekreist werden, weisen Sie das gewünschte Datum an die Eigenschaft `TodayDate` zu.

- Welche Tage im Kalenderblatt fett dargestellt werden ...
... legen Sie fest, indem Sie die gewünschten Tage als `DateTime`-Arrays an die Eigenschaften `BoldedDates`, `MonthlyBoldedDates` und `AnnuallyBoldedDates` zuweisen.

```
// Tag der Geburt fett eintragen
myMonthCalendar.BoldedDates = new DateTime[] {new DateTime(1975, 12, 26, 0,0,0,0)};

// Geburtstage zweier Personen in jedem Jahr fett eintragen
myMonthCalendar.AnnuallyBoldedDates = new DateTime[] {
    new DateTime(1975, 12, 26, 0,0,0,0),
    new DateTime(1965, 2, 26, 0,0,0,0)};
```

- Um linksseitig die Wochennummern einzublenden ...
... setzen Sie `ShowWeekNumbers` auf `true`.
- Um mehr als einen Monat anzeigen zu lassen ...
... weisen Sie `CalendarDimensions` ein `Size`-Objekt zu, mit der gewünschten Breite und Höhe in Monatsblättern.
- Um den Bereich auswählbarer Tage einzuschränken ...
... weisen Sie den Eigenschaften `MinDate` und `MaxDate` entsprechende `DateTime`-Objekte zu.

Das aktuell ausgewählte Datum bzw. den markierten Bereich können Sie von den Eigenschaften `SelectionStart` und `SelectionEnd` – oder alternativ von `SelectionRange` – ablesen.

Wenn Sie direkt auf die Auswahl eines Datums reagieren wollen, fangen Sie das `DateTimeChanged`-Ereignis ab und lesen Sie das aktuell ausgewählte Datum bzw. den markierten Bereich aus den Eigenschaften `Start` und `End` des `DateRangeEventArgs`-Parameters oder aus den Eigenschaften `SelectionStart` und `SelectionEnd` – bzw. `SelectionRange` – des Steuerelements ab.

DateTimePicker

`DateTimePicker` zeigt das eingestellte Datum übersichtlich und platz sparend in einer einzelnen Zeile an (siehe Abbildung 28.14). Möchte der Anwender ein anderes Datum auswählen, muss er den zugehörigen Kalender aufklappen oder Tag, Monat bzw. Jahr markieren und dann mithilfe des integrierten Drehfelds einstellen. Welche dieser beiden Optionen angeboten wird, legt der Programmierer über die Eigenschaft `ShowUpDown` fest. Der Standardwert `false` aktiviert die Einstellung über den aufklappbaren Kalender. Per Voreinstellung zeigt das Steuerelement anfangs das aktuelle Datum an. Sie können aber ein beliebiges Datum vorgeben, indem Sie der `Value`-Eigenschaft eine entsprechende `DateTime`-Instanz zuweisen.

```
DateTimePicker myDateTimePicker = new DateTimePicker(); // Instanziierung
myDateTimePicker.Parent = this;                          // Alternative zu this.Controls.Add(myLabel)
myDateTimePicker.Location = new Point(204, 18);

myDateTimePicker.ShowUpDown = false;                     // aufklappbares Kalenderblatt
myDateTimePicker.Value = new DateTime(1975, 12, 26);
```

In welchem Format das Datum im Steuerelement angezeigt wird, können Sie über die Eigenschaft `Format` festlegen. Mögliche Werte sind die Konstanten der Enumeration `DateTimePickerFormat`:

DateTimePickerFormat-Konstante	Beschreibung
Long	Datum im Langformat (Standard)
Short	Datum im Kurzformat
Time	Uhrzeit
Custom	Datum und Uhrzeit im Format des Formatstrings, den Sie an die CustomFormat-Eigenschaft zuweisen: <pre>myDateTimePicker.CustomFormat = "dd. MMM yyyy (hh:mm)";</pre> <pre>myDateTimePicker.Format = DateTimePickerFormat.Custom;</pre> Ausführlichere Informationen zum Aufbau von Datumsformaten finden Sie in Kapitel 25, Abschnitt »Datum und Uhrzeit«.

Tabelle 28.13 DateTimePickerFormat-Werte für die Format-Eigenschaft

HINWEIS Lang-, Kurz- und Zeitformat, wie sie über die zugehörigen DateTimePickerFormat-Werte ausgewählt werden, sind auf Betriebssystemebene definiert. Der Anwender kann die Formate über das Dialogfeld für die regionalen Einstellungen, Aufruf über die Systemsteuerung, anpassen.

Das aktuell ausgewählte Datum können Sie als DateTime-Objekt aus der Eigenschaft Value oder als String aus der Eigenschaft Text ablesen.

Wenn Sie direkt auf die Auswahl eines Datums reagieren wollen, fangen Sie eines der Ereignisse TextChanged oder ValueChanged ab.

Laufleisten

Bildlaufleisten, Schieberegler und Fortschrittsanzeigen sind sich nicht nur äußerlich sehr ähnlich. Doch trotz der visuellen und funktionellen Nähe gibt es keine echte Verwandtschaft (im Sinne einer gemeinsamen Basisklasse außer Control) und jedes der drei Steuerelemente hat sein eigenes ganz spezielles Einsatzgebiet, für das es konzipiert wurde:

- *Bildlaufleisten* sind Bedienelemente. Sie kommen meist dann zum Einsatz, wenn Inhalte angezeigt werden, die zu groß für die geplante Anzeigefläche sind. Mithilfe der Bildlaufleisten kann der Anwender dann steuern, welche Bereiche angezeigt werden.
- *Schieberegler* sind ebenfalls Bedienelemente. Sie gestatten dem Anwender, einen ganzzahligen numerischen Wert einzustellen.
- *Fortschrittsanzeigen* sind Anzeigeelemente, die den Anwender über den Fortgang einer im Hintergrund ablaufenden Aktion informieren.

Bildlaufleisten

In Windows Forms gibt es für die vertikalen und horizontalen Bildlaufleisten jeweils eigene Klassen, VScrollBar und HScrollBar, die beide auf die gemeinsame Basisklasse ScrollBar zurückgehen. Beide Klassen besitzen eine Eigenschaft Value, deren Wert intern in eine Position des Bildlauffelds in der Laufleiste umge-

rechnet wird. Umgekehrt wird, wenn der Anwender den Balken verschiebt, die neue Position in einen `int`-Wert umgerechnet und in `Value` abgespeichert.

Zur Einrichtung einer Bildlaufleiste gehört in der Regel, dass Sie festlegen,

- in welches Steuerelement die Bildlaufleiste eingebettet werden soll (Eigenschaft `Parent`),
- wie die Bildlaufleiste angedockt werden soll (Eigenschaft `Dock`),
- welcher Wertebereich kontrolliert werden soll (Eigenschaften `Minimum` und `Maximum`),
- um welchen Betrag sich der `Value`-Wert ändert soll, wenn der Anwender auf die integrierten Bildlaufpfeile klickt (Eigenschaft `SmallChange`),
- um welchen Betrag sich der `Value`-Wert ändern soll, wenn der Anwender in die Laufleiste klickt (Eigenschaft `LargeChange`),
- welche Anfangsposition das Bildlauffeld einnehmen soll (Eigenschaft `Value`),
- was beim Verschieben des Bildlauffelds geschehen soll (Ereignis `Scroll`).

ACHTUNG Der Wert von `Value` bewegt sich, wenn der Anwender das Bildlauffeld verschiebt, nicht zwischen `Minimum` und `Maximum`, sondern zwischen `Minimum` und `Maximum - LargeChange + 1`. Per Code könnten Sie auch andere Werte zuweisen, Sie sollten dies allerdings nicht tun.

HINWEIS Etliche Anzeige-Steuerelemente verfügen über vordefinierte, integrierte Bildlaufleisten (`TextBox`, `ListView`, `TreeView`). Ob diese Bildlaufleisten angezeigt werden, kann über entsprechende Eigenschaften (`Scrollable`, `ScrollBars`) festgelegt werden.

Das folgende Beispiel demonstriert die Einbettung einer vertikalen Bildlaufleiste in eine `PictureBox`. Ein Beispiel für eine `PictureBox` mit zwei nach Bedarf angezeigten Bildlaufleisten finden Sie in Kapitel 31, Abschnitt »Bilder«.

```
public partial class Form1 : Form
{
    PictureBox myPictureBox;
    VScrollBar myVScrollBar;

    public Form1()
    {
        ...

        myPictureBox = new PictureBox();
        myPictureBox.Parent = this;
        myPictureBox.Image = Image.FromFile("../..\\castle.bmp");
        myPictureBox.Location = new Point(40, 30);
        myPictureBox.Paint += new PaintEventHandler(myPictureBox_Paint);

        myVScrollBar = new VScrollBar();
        myVScrollBar.Parent = myPictureBox;
        myVScrollBar.Dock = DockStyle.Right;
        myVScrollBar.Value = 0;
        myVScrollBar.Minimum = 0;
        myVScrollBar.Maximum = myPictureBox.Image.Height - 1;
    }
}
```

```

myPictureBox.Size = new Size(myPictureBox.Image.Width + myVScrollBar.Width,
                             100);

myVScrollBar.SmallChange = 1;
myVScrollBar.LargeChange = myPictureBox.Height;
myVScrollBar.Scroll += new ScrollEventHandler(myVScrollBar_Scroll);
}

private void myVScrollBar_Scroll(Object sender, ScrollEventArgs se)
{
    myPictureBox.Refresh();
}

private void myPictureBox_Paint(object sender, PaintEventArgs e)
{
    e.Graphics.DrawImage(myPictureBox.Image,
        new Rectangle(0, 0, myPictureBox.Right - myVScrollBar.Width,
                      myPictureBox.Height),
        new Rectangle(0, myVScrollBar.Value,
                      myPictureBox.Right - myVScrollBar.Width,
                      myPictureBox.Height),
        GraphicsUnit.Pixel);
}
}

```

Um die Bildlaufleiste in die PictureBox einzubetten, wird die PictureBox-Instanz der Eigenschaft Parent zugewiesen. Anschließend wird die Bildlaufleiste rechts an die PictureBox angedockt.

Da es die Aufgabe der Bildlaufleiste ist, das in die PictureBox geladene Bild vertikal zu scrollen, wird Minimum auf 0 (vor der ersten Pixelzeile) und Maximum auf `myPictureBox.Image.Height - 1` (vor der letzten Pixelzeile) gesetzt.

Nachdem die Bildlaufleiste erzeugt und das Bild geladen wurde, kann die Größe der PictureBox festgelegt werden. Im Beispiel wird die Höhe fest vorgegeben, während die Breite so gewählt ist, dass das Bild in seiner ganzen Breite zu sehen ist.

Es ist üblich, den `LargeChange`-Bildlauf so einzustellen, dass der angezeigte Ausschnitt um eine Breite bzw. Höhe des Anzeigebereichs gescrollt wird. Im Beispiel wird dazu der `LargeChange`-Eigenschaft die Höhe der PictureBox zugewiesen.

Zu guter Letzt ist noch dafür zu sorgen, dass beim Verschieben des Bildlauffelds der angezeigte Bildbereich geändert wird. Dazu wird das `Scroll`-Ereignis behandelt. Der Code zur Anpassung des Bildausschnitts steht im vorliegenden Beispiel allerdings nicht direkt in der `Scroll`-Behandlungsmethode, sondern wurde in die Behandlungsmethode für das `Paint`-Ereignis der PictureBox ausgelagert. In der `Scroll`-Behandlungsmethode wird lediglich das Neuzeichnen der PictureBox veranlasst. Die Auslagerung des Codes in die `Paint`-Methode hat den Vorteil, dass der angezeigte Ausschnitt auch beim Wiederherstellen des Fensters mit der Position des Bildlauffelds abgestimmt wird.



Abbildung 28.15 Einsatz einer vertikalen Bildlaufleiste in einer PictureBox

HINWEIS Das Scroll-Ereignis wird nicht ausgelöst, wenn Sie per Code den Wert der Eigenschaft `Value` ändern. Um auch auf programmgesteuerte Verschiebungen reagieren zu können, müssen Sie statt `Scroll` das `ValueChanged`-Ereignis behandeln.

Schieberegler

Schieberegler sind Instanzen der Klasse `TrackBar` aus dem Namespace `System.Windows.Forms`. Schieberegler gestatten dem Anwender einen ganzzahligen, numerischen Wert einzustellen. Der eingestellte Wert wird in der Eigenschaft `Value` gespeichert. Der akzeptierte Wertebereich wird über die Eigenschaften `Minimum` und `Maximum` definiert. Über die Eigenschaft `Orientation` können Sie das Steuerelement vertikal oder horizontal einblenden.

```
myTrackBar = new TrackBar();
myTrackBar.Parent = this;
myTrackBar.Location = new Point(40, 150);

myTrackBar.Orientation = Orientation.Horizontal;
myTrackBar.Minimum = -10;
myTrackBar.Maximum = +10;
myTrackBar.Value = 0;
```

Per Voreinstellung zeigt das Steuerelement für jede Ganzzahl im Wertebereich am unteren bzw. rechten Rand einen Skalenstrich an. Wenn Sie einen etwas größeren Wertebereich einstellen, sollten Sie daher unbedingt auch die `TickFrequency` einstellen. Ansonsten riskieren Sie bei zu kurzen Schieberegler, dass die Striche zu einem schwarzen Balken verschmelzen. Wenn Sie der Eigenschaft `TickFrequency` den Wert 5 zuweisen, werden Skalenstriche für den kleinsten Wert, dann jeden 5. Wert und schließlich noch den letzten Wert eingezeichnet. Wo die Skalenstriche angezeigt werden, legen Sie über die Eigenschaft `TickStyle` fest.

TickStyle-Werte	Beschreibung
None	Keine Skalenstriche
Both	Skalenstriche auf beiden Seiten
BottomRight	Skalenstriche am unteren bzw. rechten Rand
TopLeft	Skalenstriche am oberen bzw. linken Rand

Tabelle 28.14 Werte der Enumeration `TickStyle`

Schließlich können Sie noch festlegen, um welche Beträge der Positionszeiger beim Verschieben mit der Maus (`SmallChange`) und beim Klicken in das Steuerelement (`LargeChange`) versetzt werden soll.

Den aktuell eingestellten Wert können Sie von der Eigenschaft `Value` abfragen. Soll dies zeitnah zur Änderung geschehen, behandeln Sie eines der Ereignisse `Scroll` oder `ValueChanged` (wobei Letzteres auch bei Änderungen durch den Programmcode ausgelöst wird).

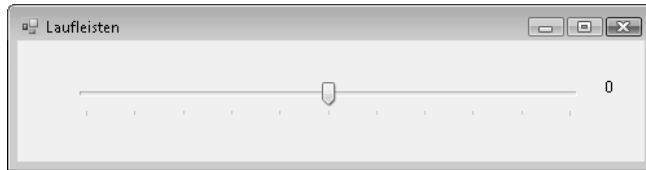


Abbildung 28.16 Schieberegler, der den Wertebereich von -10 bis +10 abdeckt und nur für jeden zweiten Wert einen Skalenstrich anzeigt. Das Label-Element rechts spiegelt den `Value`-Wert wider.

HINWEIS Der Umstand, dass `TrackBar` ebenso wie die Bildlaufleisten über die Eigenschaften `Minimum`, `Maximum` und `Value` verfügt, könnte zu der Annahme verleiten, dass auch die Beziehung zwischen `Value` und `Minimum` die gleiche ist. Dies ist aber nicht der Fall. Im Falle des Schiebereglers nimmt `Value` alle Werte von `Minimum` bis `Maximum` an.

Fortschrittsanzeige

Fortschrittsanzeigen sind Instanzen der Klasse `ProgressBar` aus dem Namespace `System.Windows.Forms`. Sie werden dazu genutzt, den Anwender über den Fortschritt länger andauernder Operationen zu informieren, auf deren Abschluss Anwendung und Benutzer warten müssen.

TIPP Wenn es Ihnen lediglich darum geht, den Anwender während der Durchführung einer längeren Operation oder Berechnung dahingehend zu beruhigen, dass das Programm noch nicht abgestürzt ist, genügt es meist, den Mauszeiger in den Wartecursor zu verwandeln. Weisen Sie dazu einfach der `Cursor`-Eigenschaft des Formulars den Wert `Cursors.WaitCursor` zu. Nach Abschluss der Operation setzen Sie den Cursor durch Zuweisung von `Cursors.Default` zurück.

TIPP Falls Sie beabsichtigen, die Fortschrittsanzeige als Füllstandsanzeige zu missbrauchen, sollten Sie wissen, dass `ProgressBar`-Instanzen immer waagrecht liegen. Ein deaktivierter Schieberegler könnte daher eine bessere Alternative sein.

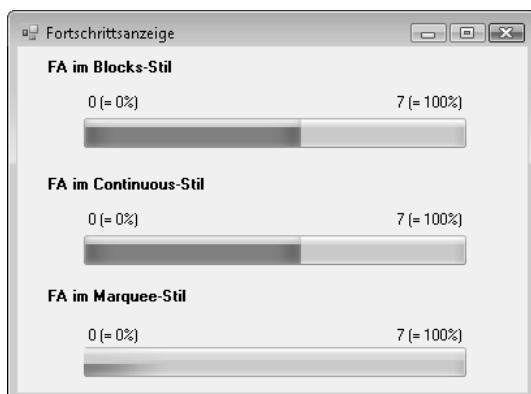


Abbildung 28.17 Fortschrittsanzeigen

ProgressBar-Fortschrittsanzeigen sind in der Regel sehr einfach einzurichten, Umrechnungen in Prozent meist nicht erforderlich (Letzteres geschieht intern im Code der ProgressBar-Klasse). Überlegen Sie sich zuerst, welche diskreten Werte den Beginn und den Abschluss der zu erledigenden Aufgabe kennzeichnen. Wenn Sie eine größere Videodatei laden, könnten Sie als Anfangswert 0 und als Endwert die Größe der Datei in KiloByte nehmen. Wenn Sie in einer Schleife n Dateien verarbeiten, wäre der Anfangswert 0 und der Endwert n . Anfangs- und Endwert weisen Sie dann den Eigenschaften `Minimum` und `Maximum` zu. (Negative Werte sind nicht erlaubt, der Wert von `Minimum` darf nicht größer als der Wert von `Maximum` sein.)

Jede `ProgressBar`-Instanz verfügt über eine Eigenschaft `Value`, deren Wert stets zwischen `Minimum` und `Maximum` liegt. (Der Standardwert ist 0. Wenn Sie `Minimum` einen Wert größer 0 zuweisen, wird `Value` automatisch mit heraufgesetzt.) Später, wenn Sie den Wert von `Value` im Zuge der durchzuführenden Aufgabe erhöhen, rechnet die `ProgressBar`-Instanz den `Value`-Wert intern in einen Prozentwert um und aktualisiert den Balken in der Fortschrittsanzeige. Wenn Sie vorhaben, den Wert von `Value` in konstanten Schritten zu erhöhen, können Sie den betreffenden Betrag in der Eigenschaft `Step` speichern.

```
ProgressBar myProgressBar = new ProgressBar(); // Instanziierung
myProgressBar.Parent = this; // Alternative zu this.Controls.Add(myProgressBar)
myProgressBar.Location = new Point(50, 100);
myProgressBar.Style = ProgressBarStyle.Continuous;

myProgressBar.Minimum = 0;
myProgressBar.Maximum = 70;
myProgressBar.Step = 1;
```

Die Klasse `ProgressBar` unterstützt drei unterschiedliche Arten von Fortschrittsanzeigen. Standard ist die Block-Anzeige. Die anderen Stile können Sie durch Zuweisung der zugehörigen `ProgressBarStyle`-Konstante an die `Style`-Eigenschaft der `ProgressBar`-Instanz auswählen.

Wert	Beschreibung
Blocks (Standard)	Der Fortschrittsbalken wird aus Blöcken fester Größe aufgebaut und nimmt nur in ganzen Blöcken zu. Wurden für die Anwendung visuelle Stile aktiviert (<code>Application.EnableVisualStyles()</code>), verwendet <code>Blocks</code> allerdings die gleiche Darstellung wie der <code>Continuous</code> -Stil.
Continuous	Der Fortschrittsbalken wird durchgehend gezeichnet und nimmt kontinuierlich zu. Wurden für die Anwendung visuelle Stile aktiviert (<code>Application.EnableVisualStyles()</code>), verwendet <code>Continuous</code> unter Windows XP allerdings die Blockdarstellung.
Marquee	Statt eines zunehmenden Fortschrittsbalkens erscheint ein Block, der wie eine Laufschrift fortwährend über das Steuerelement läuft. Die Animation ist unabhängig von den Werten für die Eigenschaften <code>Minimum</code> , <code>Maximum</code> und <code>Value</code> . Folglich ist der Anzeige auch nicht zu entnehmen, wie weit die Operation abgearbeitet bzw. wann sie beendet ist. Dieser Stil ist nur verfügbar, wenn das Betriebssystem visuelle Stile unterstützt und diese für die Anwendung aktiviert wurden (<code>Application.EnableVisualStyles()</code>).

Tabelle 28.15 Werte der Enumeration `ProgressBarStyle`

Während die auszuführende Operation abgearbeitet wird, können Sie die Fortschrittsanzeige aktualisieren, indem Sie mithilfe der Methoden `PerformStep()` oder `Increment(int n)` den Wert von `Value` erhöhen. Mit

`PerformStep()` erhöhen Sie den Wert von `Value` um den Betrag, der in der Eigenschaft `Step` gespeichert ist. Mit `Increment(int n)` können Sie `Value` um einen beliebigen Betrag erhöhen.

```
for (int i = 0; i < 7; ++i)
{
    // ... Operation durchführen
    myProgressBar.PerformStep();
}

for (int i = 0; i < 7; ++i)
{
    // ... Operation durchführen
    myProgressBar.Increment(2);
}
```

ACHTUNG Sie können den Wert der `Value`-Eigenschaft auch direkt setzen, müssen dann allerdings darauf achten, dass Sie nur Werte zwischen `Minimum` und `Maximum` zuweisen.

Listen- und Strukturansicht

Listenansicht (`ListView`) und Strukturansicht (`TreeView`) gehören zweifelsohne zu den komplexesten Komponenten, die im Repertoire der Standardsteuerelemente zu finden sind. Nichtsdestotrotz sind sie den meisten Anwendern vertraut (beispielsweise von der Arbeit mit dem Windows Explorer) und werden wegen ihrer einfachen Handhabung und übersichtlichen Form der Datenpräsentation geschätzt.

Die besondere Stärke der Listenansicht ist die Datenpräsentation in Listen- oder Tabellenform, während die Strukturansicht Daten in einer hierarchischen Ordnung anzeigt. Dabei ist es sowohl für die Listen- wie auch die Strukturansicht gänzlich unerheblich, welche Arten von Daten repräsentiert werden sollen und wie diese in der Datenquelle organisiert sind.

Diese Flexibilität hat allerdings ihren Preis. Anders als beispielsweise die `ListBox`, die beliebige Objekte präsentiert und aus den Objekten selbst die Informationen für die Präsentation im Listenfeld ableitet, arbeiten `ListView` und `TreeView` nur mit Objekten spezieller Element-Klassen zusammen (`ListViewItem` für `ListView` und `TreeNode` für `TreeView`). Diese Objekte liefern der `ListView` bzw. `TreeView` alle Informationen, die zur Präsentation und Verwaltung im Steuerelement nötig sind. Allerdings muss der Programmierer die `ListViewItem`- bzw. `TreeNode`-Objekte zuerst einmal aus seinen Originaldaten erzeugen.

Nehmen wir z. B. die vorliegende Liste von `String`-Arrays.

```
nations = new List<String[]>();
nations.Add(new String[5] { "Deutschland", "Europa", "Berlin", "3.460.000", "Hauptstadt" });
nations.Add(new String[5] { "Deutschland", "Europa", "Frankfurt", "652.000", "" });
...
nations.Add(new String[5] { "Japan", "Asien", "Tokio", "11.810.000", "Hauptstadt" });
```

Jedes `String`-Array enthält Angaben zu Land, Kontinent und den Daten einer Stadt in dem Land (Name, Einwohnerzahl, Hauptstadt). Diese Daten sollen nun so präsentiert werden, dass der Anwender sich schnell über die Städte eines bestimmten Landes informieren kann.

Es liegt nahe, zur Lösung dieser Aufgabe auf eine Kombination aus Struktur- und Listenansicht zurückzugreifen. In der Strukturansicht wählt der Anwender das Land aus, woraufhin in der Listenansicht die Städte des Landes aufgelistet werden (siehe Abbildung 28.18).

Für die gestellte Aufgabe würde es genügen, einfach der Nodes-Auflistung der Strukturansicht für jedes Land einen neuen Knoten hinzuzufügen (wobei Doppeleintragungen vermieden werden müssen). Die besonderen Vorzüge der Strukturansicht kämen dann allerdings gar nicht zum Einsatz. Besser wäre es, auf der obersten Ebene (Nodes-Auflistung der Strukturansicht) Knoten für die einzelnen Kontinente anzulegen und die Länder als untergeordnete Knoten einzutragen. Zu diesem Zweck besitzt jeder Knoten (TreeNode-Instanz) selbst auch wieder eine Nodes-Auflistung.

```
TreeView myTreeView = new TreeView();
myTreeView.Parent = this;
myTreeView.Width = 200;
myTreeView.Dock = DockStyle.Left;

// Strings mit Kontinent- und Ländernamen als Knoten erster bzw. zweiter Ordnung
// in Strukturansicht einfügen
foreach (string[] nation in nations)
{
    // Sofern der Kontinent (nation[1]) noch nicht in der obersten
    // TreeNodeCollection (myTreeView.Nodes) zu finden ist, trage ihn nun ein
    if (!myTreeView.Nodes.ContainsKey(nation[1]))
    {
        myTreeView.Nodes.Add(nation[1], nation[1]);
    }

    // Sofern das Land (nation[0]) noch nicht in der untergeordneten
    // TreeNodeCollection für den Kontinent (myTreeView.Nodes[index].Nodes)
    // zu finden ist, trage es nun ein
    int contIndex = myTreeView.Nodes.IndexOfKey(nation[1]);
    if (!myTreeView.Nodes[contIndex].Nodes.ContainsKey(nation[0]))
    {
        myTreeView.Nodes[contIndex].Nodes.Add(nation[0], nation[0], -1, -1);
    }
}
```

Für die Listenansicht werden drei Spalten definiert und die Detailansicht eingestellt:

```
ListView myListView = new ListView();
myListView.Parent = this;
myListView.BringToFront(); // Korrektur der Z-Reihenfolge ist wegen Dock.Fill
myListView.Dock = DockStyle.Fill; // nötig

myListView.Columns.Add("Stadt", 75, HorizontalAlignment.Left);
myListView.Columns.Add("Einw.", 75, HorizontalAlignment.Left);
myListView.Columns.Add("Hauptstadt", 150, HorizontalAlignment.Left);
myListView.View = View.Details;
```

Dann wird das AfterSelect-Ereignis der Strukturansicht abgefangen und in der zugehörigen Ereignisbehandlungsmethode wird das nations-Array nach Städten aus dem gewählten Land durchsucht. Für jede Stadt wird ein ListViewItem mit dem Namen der Stadt und zwei Unter-elementen für Einwohnerzahl und Hauptstadtangabe erzeugt und der Items-Auflistung der Listenansicht hinzugefügt.

```
myTreeView.AfterSelect += myTreeView_AfterSelect;
...

// Städte zu ausgewähltem Land in Liste laden
private void myTreeView_AfterSelect(Object sender, TreeViewEventArgs e)
{
    // alte Liste löschen
    myListView.Items.Clear();

    foreach (string[] nation in nations)
    {
        ListViewItem item;

        // Listenelemente für das Land aus der Strukturansicht erzeugen
        if (nation[0] == e.Node.Text)
        {
            item = new ListViewItem(nation[2]);

            // Unter-elemente für die zweite und dritte Spalte
            item.SubItems.Add(nation[3]); // Einwohnerzahl
            item.SubItems.Add(nation[4]); // Hauptstadt

            myListView.Items.Add(item);
        }
    }
}
```

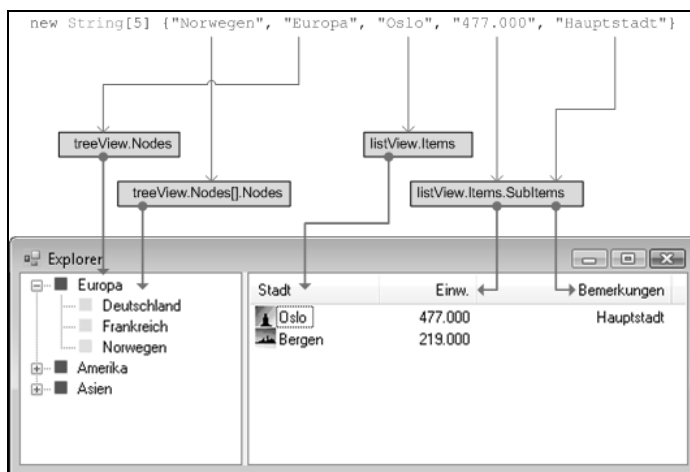


Abbildung 28.18 Struktur- und Listenansicht (Letztere in Detailansicht)

Listenansicht

Listenansichten sind Instanzen der Klasse `ListView` aus dem Namespace `System.Windows.Forms`, die Daten in Listen- oder Tabellenform präsentieren.

Listenansichten sind keine Hybridgebilde wie die `ListBox`-Listenfelder, die gleichzeitig als Container und Ansicht fungieren. Nichtsdestotrotz verfügt auch die `ListView` über eine interne Datenbasis, doch der Akzent hat sich verschoben. Während `ListBox` als Datenbasis noch die zu präsentierenden Objekte selbst speichert und versucht, aus diesen Objekten die Informationen für die Präsentation herauszuziehen, verlangt `ListView` als Datenbasis spezialisierte Objekte, deren Aufgabe ist, die Präsentationsdaten zu liefern, und die bestenfalls noch einen Verweis auf die eigentlichen Datenobjekte enthalten (sofern solche Datenobjekte überhaupt existieren).

Listenansichten unterstützen fünf verschiedene Anzeigemodi, erlauben die Sortierung und Gruppierung der anzuzeigenden Elemente, präsentieren neben Text auch gerne Bilder und können zu jedem Element Detailinfos speichern und anzeigen. Trotzdem ist die Einrichtung einer einfachen `ListView` zur Anzeige einer Reihe von Strings erstaunlich unkompliziert. Sie müssen lediglich die `ListViewItems` für die Strings erstellen und den Anzeigemodus vorgeben.

```
ListView myListView = new ListView();
myListView.Parent = this;
myListView.Location = new Point(50, 50);
myListView.Size = new Size(250, 100);

myListView.View = View.List;

myListView.Items.Add(new ListViewItem("Element 1"));
myListView.Items.Add(new ListViewItem("Element 2"));
myListView.Items.Add(new ListViewItem("Element 3"));
```

Die Elemente (ListViewItem)

Die Datenbasis der Listenansicht ist die `Items`-Auflistung. Für jeden Eintrag, der in der Listenansicht erscheinen soll, müssen Sie ein `ListViewItem`-Objekt erzeugen und dieses der `Items`-Auflistung hinzufügen. Wie viel Arbeit Sie dabei investieren, hängt davon ab, welche Ansichten und Features der Listenansicht Sie aktiv unterstützen wollen.

Reine Stringdarstellung

Im einfachsten Fall speichern Sie in den `ListViewItem`-Objekten lediglich den Anzeigetext, den Sie zu diesem Zweck als erstes Argument an den Konstruktor übergeben oder der `Text`-Eigenschaft zuweisen.

```
ListViewItem li = new ListViewItem("Element 1");
```

Darstellung mit Text und Bild

Wenn Sie die Elemente mit Text und Bild anzeigen möchten, müssen Sie die Bilder in den `ImageList`-Eigenschaften `SmallImageList` und `LargeImageList` speichern.

`SmallImageList` speichert die Bilder für die Ansichten `List`, `SmallIcon` und `Details`. `LargeImageList` bedient die verbleibenden Ansichten `LargeIcon` und `Tile`.

Die Größe der Bilder wird von der jeweiligen `ImageList`-Auflistung festgelegt. Per Voreinstellung skalieren beide Auflistungen die eingefügten Bilder auf 16 x 16 Pixel. Für die Darstellung mit großen Bildern ist dies sicherlich zu klein, weswegen Sie zumindest für `LargeImageList` andere Dimensionen wählen sollten:

```
myListView.SmallImageList = new ImageList();
myListView.LargeImageList = new ImageList();
myListView.LargeImageList.ImageSize = new Size(50, 50);

myListView.SmallImageList.Images.Add(Bitmap.FromFile("..\\..\\City.bmp"));
myListView.SmallImageList.Images.Add(Bitmap.FromFile("..\\..\\Capital.bmp"));
myListView.LargeImageList.Images.Add(Bitmap.FromFile("..\\..\\City.bmp"));
myListView.LargeImageList.Images.Add(Bitmap.FromFile("..\\..\\Capital.bmp"));
```

Da es üblich ist, für ein und dasselbe Element stets auch das gleiche Bild zu verwenden⁷, genügt es unter Umständen, die benötigten Bilder einmalig zu erstellen und dann sowohl in `SmallImageList` als auch `LargeImageList` zu speichern. Für die unterschiedlichen Abmessungen sorgt dann die Skalierung durch die `ImageList`. Sollten Sie allerdings feststellen, dass die Qualität der Bilder durch die Skalierung zu sehr leidet, bleibt Ihnen nichts anderes übrig, als zwei Bildsätze zu erstellen.

Wenn Sie anschließend die `ListViewItem`-Elemente erzeugen, übergeben Sie den nullbasierten Index des für das jeweilige Element zu verwendende Bild als zweites Konstruktorargument oder weisen Sie es der `ListViewItem`-Eigenschaft `ImageIndex` zu.

```
myListView.Items.Add(new ListViewItem("Frankfurt", 0));
myListView.Items.Add(new ListViewItem("Berlin", 1));
myListView.Items.Add(new ListViewItem("Dresden", 0));
```

HINWEIS Wenn Sie die Listenansicht nur in einer vorgegebenen Ansicht verwenden, genügt es die `ImageList` zu füllen, die von der betreffenden Ansicht verwendet wird.

Unterelemente und Spaltendefinition

Die Ansichten `Detail` und `Tile` (Kacheln) können zu jedem Element weitere Detailinformationen anzeigen. Diese Detailinformationen müssen Sie als Unterelemente (Instanzen der Klasse `ListViewSubItem`) der `SubItems`-Auflistung des Hauptelements hinzufügen.

```
ListViewItem li = new ListViewItem("Berlin");
li.SubItems.Add("3.460.000");
li.SubItems.Add("Hauptstadt");
```

Die `SubItems`-Untereinträge bestehen nur aus Text und werden nur in den Ansichten `Detail` und `Tile` angezeigt und in diesen auch nur dann, wenn entsprechende Spalten definiert sind.

Die Spalten einer Listenansicht werden in der Eigenschaft `Columns` verwaltet. Um eine neue Spalte einzurichten, rufen Sie die `Add()`-Methode auf und übergeben als Parameter die Spaltenüberschrift, die Breite in Pixeln und die Ausrichtung.

⁷ Im Beispiel werden Städte mit dem Symbol aus *City.bmp* und Hauptstädte mit dem Symbol aus *Capital.bmp* angezeigt.

```
myListView.Columns.Add("Stadt", 100, HorizontalAlignment.Left);  
myListView.Columns.Add("Einw.", 75, HorizontalAlignment.Right);  
myListView.Columns.Add("Bemerkungen", 150, HorizontalAlignment.Left);
```

- Der Text für die Spaltenüberschrift wird in der Tile-Ansicht ignoriert
- Als Breite können Sie auch die Werte -1 und -2 übergeben. Bei einem Wert von -1 passt ListView die Breite an den Inhalt der für diese Spalte vorgesehenen Daten an (sofern vorhanden). Bei einem Wert von -2 passt ListView die Breite an die Spaltenüberschrift an.
- Die Ausrichtung betrifft sowohl die Spaltenüberschrift als auch die in der Spalte angezeigten Inhalte. Mögliche Werte sind: `HorizontalAlignment.Left`, `HorizontalAlignment.Right` und `HorizontalAlignment.Center`.

ACHTUNG Die automatische Anpassung der Spaltenbreite an ihren Inhalt scheint mir recht unzuverlässig. Ich würde daher die Vorgabe fester Werte empfehlen.

HINWEIS Wenn Sie Spalten definieren, zieht die Listenansicht die Breite der ersten Spalte automatisch auch zur Dimensionierung der Elemente in den anderen Ansichten heran.

Rückverweise auf Datenquelle

Wenn Sie die Listenansicht als Benutzerschnittstelle zu einer echten Datenquelle benutzen, ist es vermutlich erforderlich, dass Sie für jedes in der Listenansicht ausgewählte Element eruieren können, welches Datenobjekt das Element repräsentiert. Zu diesem Zweck können Sie bei der Erzeugung der `ListViewItem`-Elemente in der `Tag`-Eigenschaft einen Verweis auf das originale Datenobjekt speichern.

Wenn Sie dann später über die Rückgabewerte von `SelectedIndices` oder `SelectedItem`s (bzw. `CheckedIndices` oder `CheckedItems`) ermitteln, welche Elemente ausgewählt sind, können Sie über die `Tag`-Eigenschaft der Elemente auf das zugehörige Datenobjekt zugreifen.

Aufbau im Hintergrund

Elemente, die Sie der Items-Auflistung hinzufügen, werden immer sofort gerendert, d.h., sie erscheinen direkt in der Listenansicht (immer vorausgesetzt, die Listenansicht selbst wird bereits angezeigt). Da dies viel Zeit kostet und Flackern auslösen kann, empfiehlt es sich beim Aufbau umfangreicher Items-Auflistungen, das Rendern auszuschalten.

```
// Neuzeichnen bis zum nächsten EndUpdate()-Aufruf unterdrücken  
myListView.BeginUpdate();  
  
// Hier werden die Elemente in die Items-Auflistung eingefügt  
  
myListView.EndUpdate();
```

Ansichten und weitere Konfigurationsmöglichkeiten

Es gibt eine ganze Reihe von Eigenschaften zur Konfiguration von Listenansichten, z.B.:

- `LabelEdit` – erlaubt dem Anwender, in den Text eines Elements zu klicken und den Text zu ändern
- `HideSelection` – auf `false` gesetzt, sorgt diese Eigenschaft dafür, dass ausgewählte Elemente auch dann hervorgehoben bleiben, wenn die Listenansicht den Fokus verliert
- `MultiSelect` – auf `true` gesetzt, ermöglicht diese Eigenschaft die gleichzeitige Auswahl mehrerer Elemente
- `CheckBoxes` – blendet vor jedem Element ein Kontrollkästchen ein. Über die Eigenschaften `CheckedItems` und `CheckedIndices` können Sie abfragen, für welche Elemente die Kontrollkästchen aktiviert sind. Die Kontrollkästchen sind als Alternative oder Ergänzung zur üblichen Mehrfachauswahl (`MultiSelect = true`, Abfrage über Eigenschaften `SelectedItems` und `SelectedIndices`) gedacht.

HINWEIS Die Einblendung der Kontrollkästchen kann schnell zu unerwünschten Überlappungen in der Präsentation der Elemente führen.

Etliche weitere Konfigurationsmöglichkeiten sind an die einzelnen Ansichten gekoppelt, die Sie über die Eigenschaft `View` einstellen können.

Ansicht	Beschreibung
List	Jedes Element wird mit Text und kleinem Symbol dargestellt. Die Elemente werden spaltenweise angezeigt. Relevante Eigenschaften: <code>SmallImageList</code>
SmallIcon	Jedes Element wird mit Text und kleinem Symbol dargestellt. Die Elemente werden standardmäßig zeilenweise (<code>ListViewAlignment.Top</code>) angezeigt. Wenn Sie die Eigenschaft <code>Alignment</code> auf <code>ListViewAlignment.Left</code> setzen, erhalten Sie eine List-ähnliche Darstellung. Setzen Sie <code>LabelWrap</code> auf <code>false</code> , wenn längere Texte nicht umbrochen werden sollen. Relevante Eigenschaften: <code>Alignment</code> , <code>AutoArrange</code> , <code>LabelWrap</code> , <code>SmallImageList</code> , <code>Groups</code>
LargeIcon	Wie <code>SmallIcon</code> , nur dass die Bilder aus der <code>LargeImageList</code> -Auflistung bezogen werden.
Tile	Jedes Element wird mit Text und großem Symbol dargestellt. Unterelement-Informationen werden daneben bzw. darunter eingeblendet. Die Größe der Kacheln wird nicht automatisch an die angezeigten Inhalte angepasst. Sie haben aber die Möglichkeit, genügend große Abmessungen über die <code>TileSize</code> -Eigenschaft der Listenansicht vorzugeben. Achtung! Die Eigenschaft <code>CheckBoxes</code> muss vor dem Wechsel zu dieser Ansicht auf <code>false</code> gesetzt werden. Außerdem wird diese Ansicht erst ab Windows XP/2003 unterstützt. Relevante Eigenschaften: <code>TileSize</code> , <code>ListViewItem.SubItems</code> , <code>Columns</code> , <code>LargeImageList</code> , <code>Groups</code>
Details	Jedes Element wird als eine Zeile dargestellt. Die erste Spalte zeigt den Text und das kleine Symbol. In den weiteren Spalten folgen die Unterelement-Informationen. Über die Eigenschaft <code>HeaderStyle</code> können Sie die Spaltenüberschriften ausblenden (<code>ColumnHeaderStyle.None</code>), in Schaltflächen verwandeln (<code>ColumnHeaderStyle.Clickable</code>) oder ganz normal darstellen lassen (<code>ColumnHeaderStyle.NonClickable</code>). Wenn Sie dem Anwender das Umarrangieren der Spalten gestatten möchten, setzen Sie <code>AllowColumnReorder</code> auf <code>true</code> . Wenn beim Auswählen eines Elements die komplette Zeile (mit allen Spalten) markiert werden soll, setzen Sie <code>FullRowSelect</code> auf <code>true</code> . Wenn Sie Gitterlinien anzeigen möchten, setzen Sie <code>GridLines</code> auf <code>true</code> . Relevante Eigenschaften: <code>Columns</code> , <code>HeaderStyle</code> , <code>AllowColumnReorder</code> , <code>FullRowSelect</code> , <code>GridLines</code> , <code>ListViewItem.SubItems</code> , <code>Columns</code> , <code>SmallImageList</code> , <code>Groups</code>

Tabelle 28.16 Die verschiedenen Ansichten und ihre speziellen Konfigurationsmöglichkeiten

Sortierung

Mithilfe der Eigenschaft `Sorting` können Sie die Elemente aufsteigend (`SortOrder.Ascending`) oder absteigend (`SortOrder.Descending`) sortieren lassen. Sortiert wird dabei nach dem Text des Elements (1. Spalte). Wenn Sie keine Sortierung wünschen, setzen Sie `Sorting` auf `SortOrder.None`.

```
myListView.Sorting = SortOrder.Ascending;
```

Wenn Sie möchten, dass der Anwender die Sortierreihenfolge durch Klick in eine Spaltenüberschrift umkehren kann, achten Sie darauf, dass die Eigenschaft `HeaderStyle` den Wert `ColumnHeaderStyle.Clickable` hat und fangen Sie das `ColumnClick`-Ereignis ab:

```
myListView.HeaderStyle = ColumnHeaderStyle.Clickable;  
myListView.ColumnClick += new ColumnClickEventHandler(myListView_ColumnClick);
```

In der Ereignisbehandlungsmethode ändern Sie den Wert der Eigenschaft `Sorting`.

```
private void myListView_ColumnClick(Object sender, ColumnClickEventArgs e)  
{  
    if (myListView.Sorting == SortOrder.Ascending)  
        myListView.Sorting = SortOrder.Descending;  
    else  
        myListView.Sorting = SortOrder.Ascending;  
}
```

HINWEIS Für die Sortierung wird in diesem Fall immer nur die erste Spalte herangezogen. Wenn Sie nach der Spalte sortieren möchten, die angeklickt wurde, müssen Sie über `e.Column` den Index der angeklickten Spalte abfragen und der Eigenschaft `ListViewItemSorter` ein passendes `IComparer`-Objekt zuweisen, welches nach den Werten der ausgewählten Spalte sortiert. Die Auflistung wird daraufhin automatisch neu sortiert. (Um die Sortierung explizit anzustoßen, rufen Sie die Methode `Sort()` auf.)

Gruppierung

Statt die Elemente einfach nur neben- und untereinander auflisten zu lassen, können Sie sie auch zusätzlich noch in Gruppen aufteilen. Bezüglich der Festlegung der Gruppenzugehörigkeit sind Sie gänzlich frei, d.h. Sie definieren die Gruppen und Sie legen fest, welches Element welcher Gruppe angehört.

Gruppen definieren Sie, indem Sie die Klasse `ListViewGroup` instanziiieren und das erzeugte Objekt der `Groups`-Auflistung hinzufügen. Der String, den Sie dabei dem `ListViewGroup`-Konstruktor übergeben, wird später im Listenfeld als Gruppenüberschrift angezeigt.

```
ListViewGroup g1 = new ListViewGroup("Gruppe 1");  
ListViewGroup g2 = new ListViewGroup("Gruppe 2");  
myListView.Groups.Add(g1);  
myListView.Groups.Add(g2);
```

Wenn Sie anschließend die `ListViewItem`-Objekte erzeugen, übergeben Sie die Gruppe als zweites oder drittes Argument.

```
ListViewItem item1 = new ListViewItem("Text", g1);
ListViewItem item2 = new ListViewItem("Text", imageIndex, g1);
```

Die Gruppenzugehörigkeit bereits eingefügter Elemente können Sie über deren Group-Eigenschaft ändern.

Wenn Sie die Gruppierung vorübergehend ausschalten möchten, setzen Sie die ListView-Eigenschaft ShowGroups auf false.

HINWEIS Mit Ausnahme von List unterstützen alle Ansichten die Gruppierung der angezeigten Elemente.

ACHTUNG Die Aufteilung in Gruppen wird erst ab Windows XP/2003 unterstützt und kann in bestimmten Konfigurationen zu Überlappungen führen.

Kontextmenü zur Auswahl der Ansicht

Die Listenansicht verfügt über keine vordefinierte Schnittstelle, über die der Anwender zwischen den fünf verschiedenen Anzeigemodi wechseln könnte. Dies bedeutet zum einen, dass Sie als Programmierer die gestalterische Freiheit haben, die Listenansicht in nur einer voreingestellten Ansicht zu belassen (etwa als gekachelte Liste oder als Tabelle). Andererseits heißt es auch, dass Sie die fehlende Schnittstelle einrichten müssen, wenn Sie dem Anwender die Möglichkeit zum Wechseln einräumen möchten.

Eine solche Schnittstelle – etwa in Form von Menü- oder wie nachfolgend beschrieben Kontextmenübefehlen – ist allerdings schnell erstellt, denn schließlich bedarf es zur Umschaltung zwischen den Ansichten nur einer einfachen Zuweisung an die View-Eigenschaft. Lediglich im Falle der Tile-Ansicht sollten Sie beachten, dass für diese Ansicht keine Kontrollkästchen angezeigt werden dürfen. (Und falls nicht bereits geschehen, sollten Sie die tileSize festlegen.)

Der folgende Code erzeugt ein entsprechendes Kontextmenü und verbindet es mit der Listenansicht:

```
ContextMenuStrip contextMenuStrip = new ContextMenuStrip();
contextMenuStrip.Items.AddRange(new ToolStripItem[] {
    new ToolStripMenuItem("Kleine Symbole", null, this.command_Click),
    new ToolStripMenuItem("Große Symbole", null, this.command_Click),
    new ToolStripMenuItem("Kacheln", null, this.command_Click),
    new ToolStripMenuItem("Liste", null, this.command_Click),
    new ToolStripMenuItem("Details", null, this.command_Click),
});
myListView.ContextMenuStrip = contextMenuStrip;
```

Die zugehörige Ereignisbehandlungsmethode command_Click() könnte wie folgt aussehen:

```
private void command_Click(object sender, EventArgs e)
{
    if (sender is ToolStripMenuItem)
    {
        ToolStripMenuItem mi = (ToolStripMenuItem) sender;
        if (mi.Text == "Kleine Symbole")
            myListView.View = View.SmallIcon;
        else if (mi.Text == "Große Symbole")
            myListView.View = View.LargeIcon;
        else if (mi.Text == "Kacheln")
```

```
{
    myListView.CheckBoxes = false;
    myListView.TileSize = new Size(myListView.ClientSize.Width/2, 50);
    myListView.View = View.Tile;
}
else if (mi.Text == "Liste")
    myListView.View = View.List;
else if (mi.Text == "Details")
    myListView.View = View.Details;
}
```

Ereignisbehandlung

Das wichtigste Ereignis für die Listenansicht ist `SelectedIndexChanged`, welches ausgelöst wird, wenn der Anwender ein Element auswählt. Welches Element ausgewählt ist bzw. welche Elemente in einer Listenansicht mit Mehrfachauswahl aktuell ausgewählt sind, können Sie von den Eigenschaften `SelectedIndices` und `SelectedItems` abfragen.

- `SelectedIndices` gibt eine `ListView.SelectedIndexCollection`-Auflistung der Indizes der ausgewählten Elemente zurück. Die Indizes beziehen sich auf die Position der Elemente in der Items-Auflistung.
- `SelectedItems` gibt eine `ListView.SelectedListViewItemCollection`-Auflistung der Elemente zurück.

```
private void listView1_SelectedIndexChanged(object sender, EventArgs e)
{
    string s = "Ausgewählt sind: ";
    foreach (ListViewItem li in listView1.SelectedItems)
    {
        s += li.Text + " ";
    }
    MessageBox.Show(s);
}
```

HINWEIS Das `SelectedIndexChanged`-Ereignis wird zweimal ausgelöst, wenn eine Auswahl durch eine andere Auswahl ersetzt wird.

Wenn die Elemente mit Kontrollkästchen dargestellt werden (Eigenschaft `CheckBoxes`), können Sie das `ItemChecked`-Ereignis abfangen und über die `Item`-Eigenschaft des `ItemCheckedEventArgs`-Parameters auf das Element zugreifen, dessen Kontrollkästchen gerade geändert wurde. Oder Sie fragen über `CheckedIndices` bzw. `CheckedItems` ab, für welche Elemente das Kontrollkästchen gesetzt ist.

Wenn Ihre `ListViewItem`-Elemente nur die visuelle Oberfläche zu dahinter liegenden Datenobjekten darstellen, können Sie über die `Tag`-Eigenschaft der Elemente auf das zugehörige Datenobjekt zugreifen – immer vorausgesetzt, Sie haben beim Erzeugen der `ListViewItem`-Elemente daran gedacht, den Verweis auf das Datenobjekt in der `Tag`-Eigenschaft zu speichern.

Weitere interessante Ereignisse sind z.B. `AfterLabelEdit` und `ColumnClick`.

Strukturansicht

Strukturansichten sind Instanzen der Klasse `TreeView` aus dem Namespace `System.Windows.Forms` und dienen der hierarchischen Präsentation von Daten.

So komplex die Strukturansicht auf den ersten Blick auch erscheint, die Handhabung ist erstaunlich einfach. Die einzelnen Knoten, die in der Strukturansicht angezeigt werden, sind Instanzen der Klasse `TreeNode`. Jeder Knoten besitzt eine `Text`-Eigenschaft, die den anzuzeigenden String enthält und eine `Nodes`-Auflistung mit den Verweisen auf die untergeordneten Knoten. Die Strukturansicht selbst besitzt ebenfalls eine `Nodes`-Auflistung für die Knoten der obersten Ebene (*Stammknoten*).

```
TreeView myTreeView = new TreeView();
myTreeView.Parent = this;
myTreeView.Location = new Point(50, 200);
myTreeView.Size = new Size(250, 200);

// Stammknoten mit drei untergeordneten Knoten
myTreeView.Nodes.Add("Europa");
myTreeView.Nodes[0].Nodes.Add("Deutschland");
myTreeView.Nodes[0].Nodes.Add("Frankreich");
myTreeView.Nodes[0].Nodes.Add("Norwegen");

// Weiterer Stammknoten mit zwei untergeordneten Knoten
myTreeView.Nodes.Add("Afrika");
myTreeView.Nodes[1].Nodes.Add("Senegal");
myTreeView.Nodes[1].Nodes.Add("Elfenbeinküste");
```

Konfiguration

Strukturansichten wie in Abbildung 28.18 werden meist verwendet, wenn der Anwender einen einzelnen Knoten auswählen soll und die Anwendung direkt auf die Auswahl reagiert. In Fällen wo der Anwender mehrere Knoten auswählen soll, man denke beispielsweise an die Dialogfelder zur benutzerdefinierten Installation, ist es hingegen üblich, Kontrollkästchen vor den Knoten einzublenden. Setzen Sie dazu einfach die Eigenschaft `CheckBoxes` auf `true`.

Per Voreinstellung verbinden Linien sowohl die Stammknoten untereinander als auch die übergeordneten mit den untergeordneten Knoten. Außerdem bekommen Knoten mit untergeordneten Knoten eine Schaltfläche mit Plus - bzw. Minuszeichen vorangestellt.

- Wenn Sie keine Schaltflächen wünschen, setzen Sie die Eigenschaft `ShowPlusMinus` auf `false`. Die Anwender können die Knoten dann nur durch Doppelklick oder mit den Pfeiltasten erweitern bzw. reduzieren.
- Wenn Sie keine Linien zwischen den Stammknoten wünschen, setzen Sie die Eigenschaft `ShowRootLines` auf `false`. Es werden dann allerdings auch keine Schaltflächen angezeigt.
- Wenn Sie überhaupt keine Linien möchten, setzen Sie `ShowLines` auf `false` und überlegen Sie sich, ob Sie für ausgewählte Knoten statt des reinen Knotentextes die gesamte Zeile hervorheben möchten (Eigenschaft `FullRowSelect` auf `true`).

Schließlich sei noch die Eigenschaft `LabelEdit` erwähnt, die Sie auf `true` setzen müssen, wenn Sie dem Anwender die Möglichkeit zugestehen wollen, die Knotentexte zu ändern.

Die Knotenelemente

Die Datenbasis der Strukturansicht ist die Nodes-Auflistung. Für jeden Eintrag, der in der Strukturansicht erscheinen soll, müssen Sie ein `TreeNode`-Objekt erzeugen und dieses der Nodes-Auflistung hinzufügen.

Reine Stringdarstellung

Im einfachsten Fall speichern Sie in den `TreeNode`-Objekten lediglich den Anzeigetext, den Sie zu diesem Zweck als erstes Argument an den Konstruktor übergeben oder der `Text`-Eigenschaft zuweisen.

```
TreeNode node = new TreeNode("Knoten 1");
```

Wenn Sie vermeiden möchten, dass ein Knoten mehrfach in eine Nodes-Auflistung eingetragen wird, sollten Sie jedem Knoten einen eindeutigen Namen als Schlüssel zuweisen.

```
TreeNode node = new TreeNode("Schlüssel_A", "Knoten 1");
```

Beim Einfügen der Knoten in die Nodes-Auflistung rufen Sie dann vorab die `ContainsKey()`-Methode auf, um sicherzustellen, dass der aktuell einzufügende Knoten noch nicht in der Auflistung vorhanden ist:

```
if (!myTreeView.Nodes.ContainsKey("Schlüssel_C"))
{
    myTreeView.Nodes.Add("Schlüssel_C", "Knoten 34");
}
```

Darstellung mit Text und Bild

Wenn Sie die Knoten mit Text und Bild anzeigen möchten, müssen Sie die Bilder in der Auflistungseigenschaft `ImageList` speichern. Wenn Sie für alle Knoten dasselbe Bild verwenden möchten, müssen Sie nichts weiter tun, als dieses eine Bild in die Auflistung zu laden. Es wird dann automatisch für die Darstellung der Knoten verwendet.

Wenn Sie den verschiedenen Knoten unterschiedliche Bilder zuordnen möchten, beispielsweise ein Bild für jede Ebene, müssen Sie entsprechend mehr Bilder in die Auflistung einfügen und beim Erzeugen der Knoten festlegen, welches Bild für welchen Knoten verwendet werden soll.

```
myTreeView.ImageList = new ImageList();
myTreeView.ImageList.Images.Add(Bitmap.FromFile("..\\..\\TreeLevel1.bmp"));
myTreeView.ImageList.Images.Add(Bitmap.FromFile("..\\..\\TreeLevel2.bmp"));
...
new TreeNode("Knoten 1"), 0, 0);
new TreeNode("Knoten 1_1"), 1, 1);
new TreeNode("Knoten 1_2"), 1, 1);
```

Hier bekommt der Knoten der ersten Ebene das Bild aus *TreeLevel1.bmp* zugewiesen, während die untergeordneten Knoten mit dem Bild aus *TreeLevel2.bmp* angezeigt werden.

Seltsam ist, dass der Index in die Bildauflistung zweimal angegeben werden muss. Dies liegt daran, dass die Strukturansicht für jeden Knoten zwei Bilder anzeigt: eines für die Standardanzeige und ein weiteres für den Fall, dass der Knoten ausgewählt wurde.

```
myTreeView.ImageList = new ImageList();
myTreeView.ImageList.Images.Add(Bitmap.FromFile("..\\..\\TreeLevel1.bmp"));
myTreeView.ImageList.Images.Add(Bitmap.FromFile("..\\..\\TreeLevel1_selected.bmp"));
myTreeView.ImageList.Images.Add(Bitmap.FromFile("..\\..\\TreeLevel2.bmp"));
myTreeView.ImageList.Images.Add(Bitmap.FromFile("..\\..\\TreeLevel2_selected.bmp"));
...
new TreeNode("Knoten 1"), 0, 1);
new TreeNode("Knoten 1_1"), 2, 3);
new TreeNode("Knoten 1_2"), 2, 3);
```

Wenn Sie die Bilder für einen bereits erzeugten Knoten festlegen oder ändern möchten, weisen Sie die Bildindizes den Eigenschaften `ImageIndex` und `SelectedImageIndex` zu. Wurde der Knoten noch nicht in die Nodes-Auflistung eingefügt, können Sie die Bildindizes auch an die `Add()`-Methode übergeben:

```
myTreeView.Nodes.Add("Text", "Text", 1, 1);
```

HINWEIS Die Größe der Bilder wird von der ImageList-Auflistung auf 16 x 16 Pixel skaliert. In diesem Format sollten Sie möglichst auch die Bilddateien erstellen. Wenn Sie kleinere Symbole anzeigen möchten, versuchen Sie nicht, die Skalierung zu ändern (`ImageSize`-Eigenschaft der ImageList-Auflistung), sondern erstellen Sie 16x16-Bilder mit transparentem Hintergrund.

Unterelemente

Jeder Knoten verfügt über eine Nodes-Auflistung, in der die ihm untergeordneten Knoten gespeichert werden. Die `Add()`-Methode dieser Auflistung akzeptiert sowohl komplette `TreeNode`-Instanzen als auch String- und Index-Argumente, aus denen dann ad hoc ein neuer Knoten erzeugt wird.

```
TreeNode node = new TreeNode("Deutschland");
node.Nodes.Add("Berlin"); // Knoten mit Bild 0
node.Nodes.Add("Bonn", "Bonn", 1, 1); // Knoten mit Bild 1
```

Rückverweise auf Datenquelle

Wenn Sie die Strukturansicht als Benutzerschnittstelle zu einer echten Datenquelle benutzen, ist es vermutlich erforderlich, dass Sie für jeden in der Strukturansicht ausgewählten Knoten eruieren können, welches Datenobjekt der Knoten repräsentiert. Zu diesem Zweck können Sie bei der Erzeugung der `TreeNode`-Elemente in der `Tag`-Eigenschaft einen Verweis auf das originale Datenobjekt speichern.

Wenn Sie dann später eines der `NodeXXX`- oder `AfterXXX`-Ereignisse abfangen, können Sie über den zweiten Ereignisparameter das auslösende `TreeNode`-Objekt und über dessen `Tag`-Eigenschaft das zugehörige Datenobjekt ermitteln.

Aufbau im Hintergrund

Knoten, die Sie in der Nodes-Auflistung hinzufügen, werden immer sofort gerendert, d.h., sie erscheinen direkt in der Strukturansicht (immer vorausgesetzt, die Strukturansicht selbst wird bereits angezeigt). Da dies viel Zeit kostet und Flackern auslösen kann, empfiehlt es sich, beim Aufbau umfangreicher Nodes-Auflistungen das Rendern auszuschalten.

```
// Neuzeichnen bis zum nächsten EndUpdate()-Aufruf unterdrücken
myTreeView.BeginUpdate();

// Hier werden die Knoten in die Nodes-Auflistung eingefügt

myTreeView.EndUpdate();
```

Steuern der Strukturansicht

Der Anwender kann die Knoten der Strukturansicht erweitern, reduzieren, auswählen und, wenn Kontrollkästchen angezeigt werden, auch aktivieren oder deaktivieren. Gleiches ist natürlich auch vom Programmcode aus möglich:

Aktion	TreeView-Methode/Eigenschaft	TreeNode-Methode/Eigenschaft
Knoten erweitern		void Expand()
Knoten reduzieren		void Collapse ()
Alle untergeordneten Knoten erweitern		void ExpandAll()
Alle Knoten erweitern	void ExpandAll()	
Alle Knoten reduzieren	void CollapseAll()	
Knoten umschalten		void Toggle()
Knoten aktivieren oder deaktivieren		bool Checked
Knoten auswählen	TreeNode SelectedNode	

Tabelle 28.17 Methoden zur Steuerung der Knotenhierarchie

Der folgende Code wählt den zweiten Knoten der ersten Ebene aus und erweitert ihn.

```
myTreeView.SelectedNode = myTreeView.Nodes[1];
myTreeView.SelectedNode.Expand();
```

Navigieren in der Strukturansicht

Wenn Sie eines der NodeXXX- oder AfterXXX-Ereignisse abfangen, liefert Ihnen der zweite Ereignisparameter das auslösende TreeNode-Objekt. Von diesem aus können Sie sich bei Bedarf problemlos in der Knotenhierarchie nach unten, links, rechts oder oben bewegen.

TreeNode-Eigenschaft	Ziel
NextNode	nächster Geschwisterknoten
Nodes	Auflistung der untergeordneten Knoten
Parent	übergeordneter Knoten
PrevNode	vorheriger Geschwisterknoten

Tabelle 28.18 Eigenschaften zur Navigation in der Knotenhierarchie

Ereignisbehandlung

Hinsichtlich der Ereignisbehandlung ist es natürlich von größtem Interesse zu erfahren, wenn der Anwender mit einem der Knoten interagiert. `TreeView` bietet hierfür drei ganze Sätze von Ereignissen an: einen Satz für Interaktionen mit der Maus (`NodeXXX`) und zwei Sätze von Ereignissen, die vor (`BeforeXXX`) bzw. nach (`AfterXXX`) einer bestimmten Aktion eintreten.

Die knotenspezifischen Mausereignisse `NodeMouseClicked`, `NodeMouseDoubleClick` und `NodeMouseHover` übergeben ihren Behandlungsmethoden ein Argument vom Typ `TreeNodeMouseEventArgs` bzw. `TreeNodeMouseHoverEventArgs`, dessen `Node`-Eigenschaft auf den auslösenden Knoten verweist. (Für ein Beispiel siehe den nachfolgenden Abschnitt.)

Mit den Ereignissen `BeforeLabelEdit` und `AfterLabelEdit` können Sie reagieren, wenn der Anwender den Text eines Knotens ändert (setzt voraus, dass die `TreeView`-Eigenschaft `LabelEdit` auf `true` gesetzt ist).

Die wichtigsten Ereignisse sind aber vermutlich `AfterCheck`, `AfterCollapse`, `AfterExpand` und `AfterSelect`. Diese übergeben ein Argument vom Typ `TreeViewEventArgs` an ihre Behandlungsmethode, das zwei wichtige Eigenschaften enthält:

- die `Node`-Eigenschaft mit dem Verweis auf den auslösenden Knoten
- die `Action`-Eigenschaft, deren Wert angibt, welche Aktion das Ereignis ausgelöst hat

Wert	Beschreibung
ByKeyboard	Ereignis wurde über die Tastatur ausgelöst.
ByMouse	Ereignis wurde über die Maus ausgelöst.
Collapse	Ereignis wurde durch das Reduzieren des Knotens ausgelöst.
Expand	Ereignis wurde durch das Erweitern des Knotens ausgelöst.
Unknown	Ereignis wurde durch unbekannte Aktion ausgelöst.

Tabelle 28.19 Werte der `TreeViewAction`-Enumeration

Die folgende Behandlungsmethode gehört zu dem Städte-Explorer-Beispiel aus Abbildung 28.18. Sie wird ausgelöst, wenn der Anwender in der Strukturansicht einen Länder-Knoten auswählt, und übernimmt die Aufgabe, die Listenansicht des Programms mit den zugehörigen Städtedaten zu füllen.

```
private void myTreeView_AfterSelect(Object sender, TreeViewEventArgs e)
{
    // Alte Liste löschen
    myListView.Items.Clear();

    // Durchlaufen der Datenquelle
    foreach (string[] nation in nations)
    {
        ListViewItem item;

        // Listenelemente für das Land aus der Strukturansicht erzeugen
        if (nation[0] == e.Node.Text)
        {
            item = new ListViewItem(nation[2]);
        }
    }
}
```

```
        // Unterelemente für die zweite und dritte Spalte
        item.SubItems.Add(nation[3]);    // Einwohnerzahl
        item.SubItems.Add(nation[4]);    // Hauptstadt

        myListView.Items.Add(item);
    }
}
```

Kontextmenüs für Knoten

Wie jedes Steuerelement, so erlaubt auch die Strukturansicht die Zuweisung eines Kontextmenüs. Sie müssen lediglich das Menü als Instanz der Klasse `ContextMenuStrip` erzeugen, mit `ToolStripMenuItem`-Befehlen füllen und anschließend an die `ContextMenuStrip`-Eigenschaft der Strukturansicht zuweisen.

Aufregender ist allerdings die Möglichkeit, jedem Knoten sein eigenes Kontextmenü zuzuweisen. Ein solches Kontextmenü könnte beispielsweise einen Befehl enthalten, der gezielt alle Unterknoten des angeklickten Knotens erweitert.

Das einzige Problem mit den knotenspezifischen Kontextmenüs ist, dass die Ereignisse für die Menübefehle keine Argumente übergeben, die uns den auslösenden Knoten anzeigen. Als möglichen Workaround kann man aber zusätzlich das `NodeMouseClick`-Ereignis abfangen, das vor dem Aufruf des Kontextmenüs ausgelöst wird, und in diesem den Verweis auf den aktuell angeklickten Knoten in einem Feld des Formulars speichern:

```
public partial class Form1 : Form
{
    ...
    private TreeView myTreeView = new TreeView();
    private TreeNode mcNode = null;
    ...
    myTreeView.NodeMouseClick += myTreeView_NodeMouseClick;
}
...
private void myTreeView_NodeMouseClick(Object sender,
                                       TreeNodeMouseClickEventArgs e)
{
    mcNode = e.Node;
}
```

Danach ist die Einrichtung des Kontextmenüs für die Knoten (im folgenden Beispiel wird das Kontextmenü nur den Knoten der ersten Ebene zugewiesen) nur noch Formsache:

```
public partial class Form1 : Form
{
    ...
    // Kontextmenü aufbauen
    ContextMenuStrip treeVContextMenu = new ContextMenuStrip();
    ToolStripMenuItem treeVCommand1 = new ToolStripMenuItem();
    treeVCommand1.Text = "Knoten expandieren";
    treeVCommand1.Click += new EventHandler(this.treeVCommand1_Click);
    treeVContextMenu.Items.Add(treeVCommand1);
}
```

```
// Das Kontextmenü an die Knoten der ersten Ebene zuweisen
foreach (TreeNode n in myTreeView.Nodes)
    n.ContextMenuStrip = treeVContextMenu;
}
...
private void treeVCommand1_Click(object sender, EventArgs e)
{
    if (mcNode != null)
    {
        mcNode.ExpandAll();
        myTreeView.SelectedNode = mcNode;
    }
}
```

Container-Elemente

Die meisten Container-Elemente sind recht einfach einzusetzen und wurden bereits im Kapitel 26, Abschnitt »Benutzeroberfläche, Layout« vorgestellt. Aus diesem Grund beschränken wir uns in diesem Abschnitt auf das `TabControl`-Element.

Registerkarten

Das Registersteuerelement `TabControl` aus dem Namespace `System.Windows.Forms` gestattet Ihnen, eine größere Anzahl von Steuerelementen übersichtlich und thematisch verteilt auf mehrere Registerkarten (Instanzen der Klasse `TabPage`) zu repräsentieren.

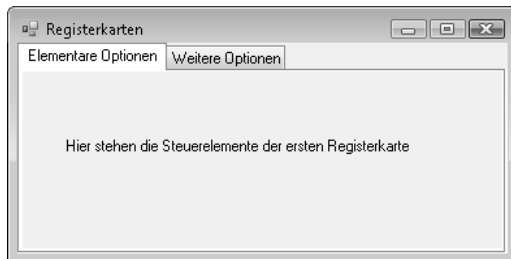


Abbildung 28.19 Registersteuerelement mit zwei Registerkarten

Wenn Sie mit Visual Studio arbeiten und eine `TabControl`-Instanz aus der Toolbox in Ihr Formular ziehen, besitzt diese automatisch zwei Registerkarten. Weitere Registerkarten können Sie über die `TabPage`-Auflistung im Eigenschaftfenster hinzufügen. Anschließend können Sie die einzelnen Registerkarten mit Steuerelementen bestücken. Um eine bestimmte Registerkarte im Designer auszuwählen, klicken Sie zuerst auf den Reiter der Registerkarte und dann in die Karte. Wenn Sie den Code selbst aufsetzen, benutzen Sie wie üblich die `Add()`-Methode, um dem Registersteuerelement die `TabPage`-Instanzen und den `TabPage`-Instanzen die Steuerelemente hinzuzufügen.

```

TabControl myTabControl = new TabControl(); // Instanziierung
myTabControl.Parent = this;                // Alternative zu this.Controls.Add(myTabControl)
myTabControl.Dock = DockStyle.Fill;

// Registerkarten anlegen
TabPage page1 = new TabPage();
TabPage page2 = new TabPage();

// Registerkarte beschriften
page1.Text = "Elementare Optionen";
page2.Text = "Weitere Optionen";

// Steuerelemente erzeugen und den Registerkarten hinzufügen
Label lb1 = new Label();
lb1.AutoSize = true;
lb1.Text = "Hier stehen die Steuerelemente der ersten Registerkarte";
lb1.Location = new Point(30,50);
page1.Controls.Add(lb1);

Label lb2 = new Label();
lb2.AutoSize = true;
lb2.Text = "Hier stehen die Steuerelemente der zweiten Registerkarte";
lb2.Location = new Point(30,50);
page2.Controls.Add(lb2);

// Registerkarten in Registersteuerelement einfügen
myTabControl.Controls.Add(page1);
myTabControl.Controls.Add(page2);

```

Konfiguration

Registerkarten füllen typischerweise den gesamten Clientbereich des übergeordneten Fensters oder Dialogs aus. Setzen Sie dazu die Dock-Eigenschaft des Registersteuerelements auf `DockStyle.Fill`.

Über die Eigenschaft `SizeMode` können Sie die Breite der Registerkarte festlegen.

Wert	Beschreibung
Normal (Standard)	Die Breite der Registerkarte richtet sich nach dem enthaltenen Text.
Fixed	Alle Registerkarten haben die gleiche Breite.
FillToRight	Die Breiten der Registerkarten einer Zeile werden so angepasst, dass sie die Gesamtbreite des Registersteuerelements ausfüllen.

Tabelle 28.20 Werte der Enumeration `TabSizeMode`

Registerkartenwechsel

Mithilfe der überladenen Methode `SelectTab()` können Sie vom Programm aus eine Registerkarte auswählen. Als Argument können Sie wahlweise den Index der Registerkarte in der `TabPage`-Auflistung, einen String mit dem Namen der Registerkarte oder eine `TabPage`-Referenz der Registerkarte übergeben.

Falls Sie darauf reagieren möchten, wenn der Anwender die Registerkarten wechselt, fangen Sie eines oder mehrere der Ereignisse `Deselecting`, `Deselected`, `Selecting` und `Selected` ab.

Menü-, Symbol- und Statusleiste

Menü- und Symbolleiste wurden bereits zu DOS-Zeiten als exzellentes Mittel geschätzt, um eine große Zahl von Programmfunktionen übersichtlich und schnell erreichbar zu präsentieren. Erscheinungsbild und Möglichkeiten der Menü- und Symbolleisten haben sich seitdem vielfach gewandelt, aber das Grundprinzip ist geblieben.

Menüleisten

Menüleisten werden in Windows Forms als Instanzen der Klasse `MenuStrip` aus dem Namespace `System.Windows.Forms` erzeugt. Die `MenuStrip`-Instanz ist allerdings nur der Container für die eigentlichen Menüs und Menübefehle, die allesamt Instanzen der Klasse `ToolStripMenuItem` sind.

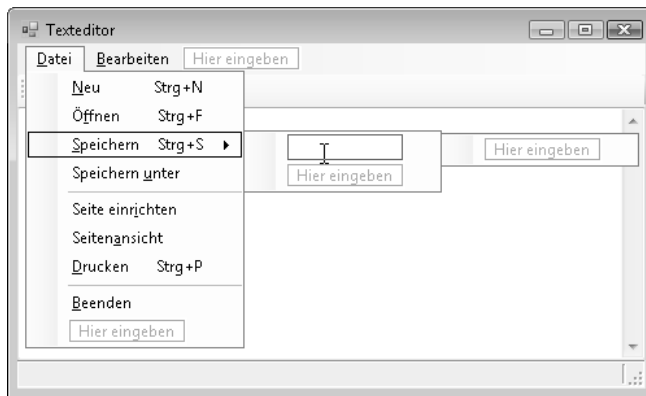


Abbildung 28.20 Menüsystem im Windows Forms-Designer

Der Aufbau des Menüsystems beginnt mit der Menüleiste, die als Instanz der Klasse `MenuStrip` erzeugt wird und üblicherweise formatfüllend in den oberen Rahmen des Formulars eingefügt wird (`DockStyle.Top`).

Die Menüs werden als Instanzen der Klasse `ToolStripMenuItem` erzeugt, wobei der Menütitel direkt dem Konstruktor übergeben werden kann.

Die Menübefehle für jedes Menü werden ebenfalls als Instanzen der Klasse `ToolStripMenuItem` erzeugt, wobei allerdings neben dem Titel auch eine Behandlungsmethode für das `Click`-Ereignis vorgesehen werden muss. Wenn Sie möchten, können Sie den Namen der Behandlungsmethode direkt an den Konstruktor übergeben – als drittes Argument (das zweite Argument ist für ein optionales Symbol gedacht; übergeben Sie einfach `null`, wenn kein Symbol angezeigt werden soll).

Die fertigen Menübefehle fügen Sie der `DropDownItems`-Collection des zugehörigen Menüs hinzu. Die fertigen Menüs fügen Sie der `Items`-Collection der Menüleiste hinzu.

Der letzte Schritt ist die Implementierung der Behandlungsmethoden:

```
// Menüleiste erzeugen und in Formular einfügen
MenuStrip myMenuStrip = new MenuStrip();
myMenuStrip.Parent = this;
myMenuStrip.Dock = DockStyle.Top;
```



```
// Einzelnes Menü erstellen und mit Menübefehlen füllen
ToolStripMenuItem file = new ToolStripMenuItem("Datei");
ToolStripMenuItem fileNew = new ToolStripMenuItem("Neu", null, fileNew_Click);
ToolStripMenuItem fileOpen = new ToolStripMenuItem("Öffnen", null, fileOpen_Click);
ToolStripMenuItem fileSave = new ToolStripMenuItem("Speichern", null, fileSave_Click);
ToolStripMenuItem fileQuit = new ToolStripMenuItem("Beenden", null, fileQuit_Click);

file.DropDownItems.AddRange(new ToolStripItem[] {fileNew, fileOpen, fileSave,
                                                new ToolStripSeparator(),
                                                fileQuit});

// Menü(s) in Menüleiste einfügen
myMenuStrip.Items.Add(file);

// Menüleiste in Formular einfügen
Controls.Add(myMenuStrip); // alternativ Zuweisung an die MainMenuStrip-Eigenschaft
                          // oder Einfügen in die Controls-Auflistung eines über-
                          // geordneten ToolStripContainer-Panels

...

// Die Behandlungsmethoden
private void fileNew_Click(object sender, EventArgs e)
{
    MessageBox.Show("Neu");
}
...
private void fileQuit_Click(object sender, EventArgs e)
{
    Application.Exit();
}
```

HINWEIS Was ein Menü von einem einfachen Menübefehl unterscheidet, ist die `DropDownItems`-Auflistung, die für einfache Menübefehle leer ist.

HINWEIS Trennelemente sind Instanzen der Klasse `ToolStripSeparator` und können sowohl in die Menüleiste (vertikaler Trennstrich) als auch in die einzelnen Menüs (horizontaler Trennstrich) eingefügt werden.

Menüs im Windows Forms-Designer

Komfortabler ist der Aufbau eines Menüsystems im Windows Forms-Designer.

1. Ziehen Sie ein `MenuStrip`-Steuerelement aus der Toolbox in Ihr Formular.

Der Designer blendet jetzt eine editierbare Menüleiste unter der Titelleiste des Fensters ein. Das Symbol für die Menüleisten-Komponente wird am unteren Rand des Windows Forms-Designers eingeblendet.

2. Legen Sie das erste Menü der Menüleiste an. Klicken Sie dazu in das Eingabefeld mit dem Hinweis *Hier eingeben* und geben Sie den Titel für das neue Menü ein.

Unterhalb und rechts neben dem Menü erscheinen daraufhin weitere Schablonen. Nach unten geben Sie die Namen der Menübefehle ein, nach rechts legen Sie weitere Menüs an.

3. Bauen Sie, wie oben beschrieben, das komplette Menüsystem auf, wobei Sie das Eigenschaftsfenster dazu nutzen können, die einzelnen Menüelemente weiter zu konfigurieren (Tastenkombinationen, Aktivierung etc.).

4. Definieren Sie die Behandlungsmethoden für die Menübefehle. Doppelklicken Sie dazu einfach auf die Menübefehle oder wählen Sie die Menübefehle durch einfachen Klick aus und richten Sie die Click-Behandlungsmethode über das Eigenschaftenfenster ein.

Soweit die grundlegende Vorgehensweise. Richtig schnell und effizient wird die Arbeit mit dem Menüeditor aber erst, wenn man auch die kleinen Tricks des Editors kennt.

Aufgabe	Ausführung
Trennelement einfügen	Legen Sie ein neues Menüelement an und geben Sie als Text den einfachen Bindestrich – ein. Trennelemente zwischen den Menüs der Menüleiste können Sie nur über den Code definieren.
Untermenü erstellen	Legen Sie ein normales Menüelement an oder wählen Sie das Menüelement aus, für das Sie ein Untermenü erstellen möchten. Wenn im Elementtitel der schwarze Pfeil erscheint, klicken Sie in die rechts anschließende Schablone und geben Sie den ersten Befehl des Untermenüs ein.
Menüelemente umarrangieren	Ziehen und verschieben Sie die Elemente einfach mit der Maus.
Alt-Tasten definieren	Setzen Sie im Titel einfach ein &-Zeichen vor das Zeichen für die Tastenkombination, z. B.: &Neu.
Click-Ereignis behandeln	Doppelklicken Sie auf das Menüelement, für das Sie eine Click-Behandlungsmethode schreiben wollen.
Erweiterte Konfiguration	Klicken Sie mit der rechten Maustaste auf das Menüelement, um das Kontextmenü des Editors zu öffnen, oder konfigurieren Sie das ausgewählte Menüelement im Eigenschaftenfenster.

Tabelle 28.21 Tipps für die Bedienung des Menüeditors

TIPP Wenn Ihnen der Menüeditor nicht zusagt, können Sie Ihre Menüsysteme auch mit dem Auflistungseditor erstellen. Wählen Sie dazu im Designer die MenuStrip-Instanz aus und öffnen Sie im Eigenschaftenfenster den Dialog zum Items-Menü. Im Fenster des Auflistungseditors können Sie dann die Menüs schnell und effizient anlegen und – mit Ausnahme der Ereignisse – konfigurieren. Die Menüelemente für die einzelnen Menüs können Sie analog bearbeiten, indem Sie den Dialog zur DropDownItems-Eigenschaft aufrufen. (Sie müssen dazu nicht erst zurück in den Designer oder zum Eigenschaftenfenster wechseln. Wählen Sie das zu bearbeitende Menü im Auflistungseditor für die Items-Eigenschaft aus, scrollen Sie bis zur DropDownItems-Eigenschaft des Menüs und öffnen Sie von hier aus den Editor.)

Die verschiedenen Menüelemente

Die wichtigsten Menüelemente sind selbstredend die einfachen Menübefehle, die der Anwender anklickt, um eine bestimmte Aktion auszulösen. Es gibt aber auch noch andere Menüelemente.

Menüelemente	Klasse	Beschreibung
Einfacher Menübefehl	ToolStripMenuItem	Das Standard-Menüelement. Der Anwender klickt auf das Menüelement, um eine bestimmte Aktion auszulösen. Wenn der Menübefehl ein Dialogfeld aufruft, sollten Sie den Titel mit Auslassungspunkten ("Neu...") abschließen. Als Ereignisbehandlung bearbeiten Sie das Click-Ereignis.
Menübefehl mit Untermenü	ToolStripMenuItem	Menüelement, zu dem ein Untermenü aufspringt, wenn der Anwender das Menüelement auswählt oder mit der Maus darüber fährt. Menüelemente, die in ihrer DropDownItems-Auflistung untergeordnete Menüelemente enthalten, werden automatisch zu Elementen mit Untermenüs. Ein weiteres Zutun des Programmierers ist nicht erforderlich. ►

Menüelemente	Klasse	Beschreibung
Menüoptionen	ToolStripMenuItem	Menüelement, dessen Zustand zwischen aktiviert (Darstellung mit Häkchen) und deaktiviert wechselt. Um eine Menüoption zu erzeugen, setzen Sie die Eigenschaft <code>CheckOnClick</code> auf <code>true</code> . Soll die Menüoption anfänglich aktiviert sein, setzen Sie die Eigenschaft <code>Checked</code> ebenfalls auf <code>true</code> . Als Ereignisbehandlung bearbeiten Sie das <code>CheckedChanged</code> -Ereignis.
Trennelemente	ToolStripSeparator	Trennelement
Textfelder	ToolStripTextBox	Texteingabefeld als Menüelement. Als Ereignisbehandlung bearbeiten Sie eines der Ereignisse <code>TextChanged</code> oder <code>Validating</code> .
Kombinationsfelder	ToolStripComboBox	Kombinationsfeld als Menüelement. Die Elemente für die aufklappbare Liste weisen Sie der Eigenschaft <code>Items</code> zu. Den Inhalt des Eingabefelds können Sie über die <code>Text</code> -Eigenschaft setzen und abfragen. Als Ereignisbehandlung bearbeiten Sie eines der Ereignisse <code>TextChanged</code> , <code>Validating</code> oder <code>SelectedIndexChanged</code> .

Tabelle 28.22 Mögliche Menüelemente

Symbole

Wie bereits angekündigt, können Sie in den Menüelementen neben dem Titel auch Symbole anzeigen. Wenn Ihre Anwendung neben einer Menüleiste auch über eine oder mehrere Symbolleisten verfügt, ist es z.B. eine nette Aufmerksamkeit, wenn Sie die Symbole der Symbolleistenschaltflächen auch in den Titeln der zugehörigen Menübefehle einblenden.

Im Windows Forms-Designer klicken Sie mit der rechten Maustaste auf das betreffende Menüelement und rufen im Kontextmenü den Befehl *Bild festlegen* auf. In dem aufspringenden Dialogfeld können Sie die Bilddatei des Symbols als *Lokale Ressource* von der Festplatte oder, falls Sie die Bilddatei schon in die Ressourcendatei der Anwendung importiert haben (siehe Kapitel 26, Abschnitt »Benutzeroberfläche, Ressourcen«), aus der *Projektressourcendatei* importieren.

Gleiches ist aber natürlich auch per Code möglich:

```
newToolStripMenuItem.Image = Bitmap.FromFile(@"../../NewDocument.bmp");
newToolStripMenuItem.Image = WindowsApplication3.Properties.Resources.NewDocument;
```

HINWEIS Wenn der Hintergrund des Symbols transparent sein soll, weisen Sie die Hintergrundfarbe des Symbols der Menüelement-Eigenschaft `ImageTransparentColor` zu.

Menüwechsel und Befehlsaktivierung

Als Anwender gehen Sie grundsätzlich davon aus, dass Sie jeden Menübefehl, der Ihnen angeboten wird, auch aufrufen können. Doch nicht jeder Menübefehl ist jederzeit sinnvoll anwendbar.

Manche Menübefehle sollten beispielsweise nur dann verfügbar sein, wenn die Anwendung in einen bestimmten Bearbeitungszustand versetzt wurde oder ein bestimmtes Unterfenster eingeblendet wurde (man denke beispielsweise an die vielen Visual Studio-Menüs, die erst angezeigt werden, nachdem ein Projekt geladen wurde). In solchen Fällen ist es empfehlenswert, die Menübefehle in eigenen Menüs zu bündeln und diese nach Bedarf der Menüleiste hinzuzufügen oder aus dieser zu entfernen.

```
// der Menüleiste ein Bearbeiten-Menü hinzufügen  
myMenuStrip.Items.Add(bearbeitenToolStripMenuItem);  
// das Bearbeiten-Menü aus der Menüleiste entfernen  
myMenuStrip.Items.Remove(bearbeitenToolStripMenuItem);
```

Ein anderer Fall sind Menübefehle, die nur dann sinnvoll anwendbar sind, wenn bestimmte Voraussetzungen gegeben sind – etwa der Befehl zum Kopieren in die Zwischenablage, der nur dann sinnvoll ist, wenn etwas zum Kopieren markiert wurde.

Solche Befehle sollten Sie nach Möglichkeit deaktivieren (Enabled-Eigenschaft auf `false`), wenn die Voraussetzungen nicht gegeben sind. Wenn wir uns in einem späteren Kapitel mit der Zwischenablage beschäftigen, werden Sie hierzu noch ein Beispiel sehen.

HINWEIS

Wenn Ihnen der Aufwand für die Deaktivierung und Aktivierung zu groß sein sollte, achten Sie zumindest darauf, dass die Behandlungsmethoden für die betreffenden Befehle so implementiert sind, dass ein Aufruf trotz nicht gegebener Voraussetzungen nicht zu Fehlern in der Anwendung führt.

Tastenkombinationen

Schnellzugriffstasten für Menübefehle definieren Sie, indem Sie dem Buchstaben, der in Kombination mit der **Alt**-Taste den Menübefehl aufrufen soll, im Titel des Menüelements ein kaufmännisches Und voranstellen, beispielsweise: `&Datei`.

Wenn Sie für die hinter dem Menübefehl stehende Aktion (Click-Ereignis) eine Tastenkombination definieren möchten, müssen Sie die gewünschte Kombination der Eigenschaft `ShortcutKeys` zuweisen:

```
einToolStripMenuItem.ShortcutKeys = ((Keys)((Keys.Control | Keys.D1))); // Strg+1
```

Wenn Sie zudem die Eigenschaft `ShowShortcutKeys` auf `true` setzen, wird die Tastenkombination automatisch in dem Titel des Menüelements eingeblendet.

```
einToolStripMenuItem.ShowShortcutKeys = true;
```

Kontextmenüs

Kontextmenüs sind kontextspezifische Menüs, die eingeblendet werden, wenn der Anwender mit der rechten Maustaste auf ein bestimmtes Steuerelement klickt. Idealerweise sollte das aufspringende Kontextmenü steuerelement- oder situationsspezifische Befehle anbieten.

Für die Definition von Kontextmenüs sieht Windows Forms die Klasse `ContextMenuStrip` vor. Instanzen der Klasse `ContextMenuStrip` besitzen eine `Items`-Eigenschaft, der Sie die Befehle des Kontextmenüs zuweisen. Das

fertige Kontextmenü müssen Sie anschließend nur noch der ContextMenuStrip-Eigenschaft des zugehörigen Steuerelements zuweisen:

```
ContextMenuStrip contextMenuStrip = new ContextMenuStrip();
contextMenuStrip.Items.AddRange(new ToolStripItem[] {
    new ToolStripMenuItem("Befehl 1", null, this.command_Click),
    new ToolStripMenuItem("Befehl 2", null, this.command_Click),
    new ToolStripMenuItem("Befehl 3", null, this.command_Click),
});
myControl.ContextMenuStrip = contextMenuStrip;

...

// Ereignisbehandlungsmethoden
// (hier wird nur eine Methode für alle Menübefehle definiert)
private void command_Click(object sender, EventArgs e)
{
    if (sender is ToolStripMenuItem)
    {
        ToolStripMenuItem mi = (ToolStripMenuItem) sender;

        if (mi.Text == "Befehl 1")
            // Handlungscode
        else if (mi.Text == "Befehl 2")
            // Handlungscode
        else if (mi.Text == "Befehl 3")
            // Handlungscode
    }
}
```

Symbolleisten

Symbolleisten, die in der Windows Forms-Programmierung als Instanzen der ToolStrip-Komponente erzeugt werden, sind in gewisser Hinsicht das grafische Analogon zu den Tastenkombinationen. Der Anwender soll eine Funktion direkt per Mausklick auswählen können, ohne langwierig durch das Menüsystem zu navigieren und sich dabei in diversen Menüs und Untermenüs zu verirren.

Der Aufbau einer Symbolleiste erfolgt ganz analog zum Aufbau einer Menüleiste, nur dass Sie statt einer MenuStrip- eine ToolStrip-Komponente verwenden und diese mit Schaltflächen (ToolStripButton-Instanzen) statt mit Menüelementen füllen.

```
// Symbolleiste erzeugen und in Formular einfügen
myToolStrip = new ToolStrip();
myToolStrip.Dock = DockStyle.Top;

// Symbolleiste mit Symbolschaltflächen füllen
ToolStripButton btnFileNew = new ToolStripButton();
    btnFileNew.Image = PROJEKTNAME.Properties.Resources.NewDocument;
    btnFileNew.ImageTransparentColor = Color.Magenta;
    btnFileNew.Click += fileNew_Click;
ToolStripButton btnFileOpen = new ToolStripButton();
```

```

        btnFileOpen.Image = PROJEKTNAME.Properties.Resources.OpenFolder;
        btnFileOpen.ImageTransparentColor = Color.Magenta;
        btnFileOpen.Click += fileOpen_Click;
        ToolStripButton btnFileSave = new ToolStripButton();
        btnFileSave.Image = PROJEKTNAME.Properties.Resources.Save;
        btnFileSave.ImageTransparentColor = Color.Magenta;
        btnFileSave.Click += fileSave_Click;

myToolStrip.Items.AddRange(new ToolStripItem[] { btnFileNew, btnFileOpen,
                                                btnFileSave });

// Symbolleiste in Formular einfügen (vor Menüleiste!)
this.Controls.Add(myToolStrip);
this.Controls.Add(myMenuStrip);

...

// Die Behandlungsmethoden
private void fileNew_Click(object sender, EventArgs e)
{
    MessageBox.Show("Neu");
}
...
private void fileQuit_Click(object sender, EventArgs e)
{
    Application.Exit();
}

```

Für Symbolschaltflächen, die mit Menübefehlen korrespondieren, werden grundsätzlich keine eigenen Click-Behandlungsmethoden definiert. Stattdessen weisen Sie dem Click-Ereignis einfach die Behandlungsmethode des zugehörigen Menübefehls zu.

ACHTUNG Wenn Sie Menüleiste und Symbolleiste in den oberen Rand des Formulars andocken, sollte die Menüleiste über der Symbolleiste stehen. Dazu ist es nötig, dass die Menüleiste in der Z-Reihenfolge nach der Symbolleiste kommt. Erzeugen Sie also zuerst die diversen Leisten, ohne die Parent-Eigenschaft zu definieren, und fügen Sie die Leisten dann anschließend in der gewünschten Reihenfolge dem Formular zu (durch Setzen der Parent-Eigenschaft oder mit der Add()-Methode der Controls-Eigenschaft).

Symbolleisten im Windows Forms-Designer

Komfortabler ist der Aufbau einer Symbolleiste im Windows Forms-Designer.

1. Ziehen Sie ein ToolStrip-Steuerelement aus der Toolbox in Ihr Formular.

Der Designer blendet jetzt eine editierbare Symbolleiste am oberen Rand des Fensters an. Das Symbol für die Symbolleisten-Komponente wird am unteren Rand des Windows Forms-Designers eingeblendet. Wenn Ihre Anwendung auch über eine Menüleiste verfügt und die Symbolleiste über der Menüleiste angezeigt wird, markieren Sie die Symbolleiste und rufen Sie den Befehl *Format/Reihenfolge/In den Vordergrund* auf.

2. Legen Sie das erste Element der Symbolleiste an. Klicken Sie dazu einfach auf die Schablone.

Der Designer fügt eine Symbolschaltfläche ein und die Schablone rückt nach rechts. Der Fokus verbleibt auf der Symbolschaltfläche.

3. Wechseln Sie in das Eigenschaftenfenster und weisen Sie der Schaltfläche ihr Symbol zu (Image-Eigenschaft, vgl. die Abschnitte »Menüleisten« und »Symbole«). Setzen Sie gegebenenfalls auch die Eigenschaften `ImageAlign` und `ImageTransparentColor`.
4. Wiederholen Sie Schritt 2 und 3, um weitere Symbolschaltflächen einzufügen.
5. Definieren Sie die Behandlungsmethoden für die Symbolleistenschaltflächen. Doppelklicken Sie dazu einfach auf die Symbolschaltflächen oder wählen Sie eine Symbolschaltfläche durch einfachen Klick aus und richten Sie die `Click`-Behandlungsmethode über das Eigenschaftenfenster ein.

TIPP Grafisch nicht so ansprechend, in der Handhabung aber vermutlich noch effizienter ist die Erstellung – oder Nachbearbeitung – im Auflistungseditor. Um den Editor zu öffnen, wählen Sie die `ToolStrip`-Instanz im Designer aus und klicken Sie im Eigenschaftenfenster auf die Schaltfläche zum `Items`-Menü. Im Fenster des Auflistungseditors können Sie dann Symbolschaltflächen schnell und effizient anlegen, umordnen und – mit Ausnahme der Ereignisse – konfigurieren.

TIPP Und noch ein Tipp: Die Symbolschaltflächen für die klassischen Befehle des *Datei-* und *Bearbeiten*-Menüs (Neu, Öffnen, Speichern, Drucken, Ausschneiden, Kopieren, Einfügen) können Sie Ihrer Symbolleiste mit nur einem Klick hinzufügen. Der zugehörige Befehl lautet *Standardelemente einfügen* und befindet sich im Aufgabenmenü der Symbolleiste im Windows Forms-Designer (Klick auf den kleinen Pfeil) wie auch im Kontextmenü der im unteren Bereich angezeigten Symbolleisten-Instanz.

Symbole

Bilder für Symbolleistenschaltflächen sind meist Bitmaps (Dateiendung *.bmp*), typischerweise in einer Größe von 25×25 Pixeln.

Im Windows Forms-Designer weisen Sie das Bild zu, indem Sie mit der rechten Maustaste auf die betreffende Symbolschaltfläche klicken und im Kontextmenü den Befehl *Bild festlegen* aufrufen. In dem aufspringenden Dialogfeld können Sie die Bilddatei des Symbols als *Lokale Ressource* von der Festplatte oder, falls Sie die Bilddatei schon in die Ressourcendatei der Anwendung importiert haben (siehe Kapitel 26, Abschnitt »Benutzeroberfläche« und »Ressourcen«), aus der *Projektr Ressourcendatei* importieren.

Gleiches ist aber natürlich auch per Code möglich:

```
// Zuweisung über Konstruktor und Laden aus Bilddatei auf Festplatte
ToolStripButton btnFileNew = new ToolStripButton(null,
    Bitmap.FromFile(@"../../NewDocument.bmp"),
    fileNew_Click);

// Zuweisung an Image-Eigenschaft und Laden aus Ressourcendatei ToolStripButton
btnFileNew.Image = PROJEKTNAME.Properties.Resources.NewDocument;
```

HINWEIS Wenn der Hintergrund des Symbols transparent sein soll, weisen Sie die Hintergrundfarbe des Symbols der `Symbolleistenelement-Eigenschaft ImageTransparentColor` zu.

QuickInfo

Die Elemente der Symbolleisten verfügen automatisch über `QuickInfo`-Fenster, die eingeblendet werden, wenn der Anwender die Maus für kurze Zeit über einem der Elemente stehen lässt. Standardmäßig wird in

diesen QuickInfo-Fenstern der vom Designer vergebene Variablenname angezeigt. Da dieser natürlich für den Anwender keine große Hilfe darstellt, sollten Sie den QuickInfo-Text durch einen kurzen, aussagekräftigen Hinweis auf die Funktion des Symbolleisten-Elements ersetzen. Den QuickInfo-Text setzen Sie über die `ToolTipText`-Eigenschaft.

Die verschiedenen Symbolleistenelemente

Neben den typischen Symbolschaltflächen können Sie noch etliche weitere Elemente in Ihre Symbolleisten einfügen:

- Symbolschaltflächen – als Instanzen der Klasse `ToolStripButton`
- Schaltflächen – ebenfalls als Instanzen der Klasse `ToolStripButton`, nur dass Sie anstelle oder zusätzlich zu dem Symbol einen Schaltflächentitel festlegen (Eigenschaften `Text`, `TextAlign`, `TextDirection` und `TextImageRelation`)
- Label – als Instanzen der Klasse `ToolStripLabel`
- Symbolschaltflächen mit aufklappbarer Liste – als Instanzen der Klasse `ToolStripSplitButton` oder `ToolStripDropDownButton`
- Textfelder – als Instanzen der Klasse `ToolStripTextBox`
- Kombinationsfelder – als Instanzen der Klasse `ToolStripComboBox`
- Fortschrittsanzeige – als Instanzen der Klasse `ToolStripProgressBar`
- Trennelemente – als Instanzen der Klasse `ToolStripSeparator`

Symbolleisten-Container und andockbare Symbolleisten

Der einfachste Weg zu beliebig andockbaren Symbolleisten führt über die Komponente `ToolStripContainer`. Betten Sie dazu eine Instanz dieser Komponente in Ihr Formular ein und lassen Sie die Komponente das Formular ausfüllen (`DockStyle.Fill`). Alle Symbol-, Menü- und Statusleisten, die Sie jetzt statt in das Formular in den Symbolleisten-Container einbetten, können vom Anwender mit der Maus aufgenommen und an einer beliebigen Seite andockt werden.

Der Symbolleisten-Container definiert vier Bereiche, die durch die Eigenschaften `TopToolStripPanelVisible`, `BottomToolStripPanelVisible`, `LeftToolStripPanelVisible` und `RightToolStripPanelVisible` repräsentiert werden. Deren `Controls`-Auflistung können Sie die Symbolleisten hinzufügen.

```
ToolStripContainer myToolStripContainer = new ToolStripContainer();
myToolStripContainer.Parent = this;
myToolStripContainer.Dock = DockStyle.Fill;

myToolStripContainer.TopToolStripPanel.Controls.Add(myToolStrip);
myToolStripContainer.TopToolStripPanel.Controls.Add(myMenuStrip);
```

TIPP

Im Windows Forms-Designer können Sie bereits eingerichtete Symbolleisten besonders einfach nachträglich andockbar machen. Ziehen Sie eine `ToolStripContainer`-Komponente in Ihr Formular, klicken Sie auf den Link *Ausfüllformular andocken* und anschließend auf *Steuerelemente neu unterordnen*. Sollte das Ergebnis nicht Ihren Vorstellungen entsprechen, nehmen Sie die Aktionen zurück (`[Strg] + [Z]`). (Achtung! Wenn Sie den Container löschen, werden auch die eingefügten Symbolleisten entfernt.)

HINWEIS Damit eine Symbol-, Menü- oder Statusleiste vom Anwender verschoben werden kann, muss sie einen sichtbaren Griff haben (Eigenschaft `GripStyle` auf `ToolStripGripStyle.Visible` setzen).

Statusleisten

Statusleisten sind Instanzen der Klasse `StatusStrip` aus dem Namespace `System.Windows.Forms`. Statusleisten werden üblicherweise an den unteren Rand des Formulars andockt und zur Einblendung von Hinweistexten und Statusinformationen benutzt.

Für die verschiedenen Informationen, die Sie einblenden möchten, müssen Sie eigene Anzeigebereiche in Form von `ToolStripStatusLabel`-Instanzen einrichten. Neben den üblichen Label-Feldern können Sie aber auch Fortschrittsanzeigen (`ToolStripProgressBar`) oder Schaltflächen mit aufklappbarer Liste (`ToolStripDropDownButton` oder `ToolStripSplitButton`) einfügen.

Die Größe der Anzeigebereiche können Sie über die `Size`-Eigenschaft definieren oder Sie passen den Anzeigebereich an den anzuzeigenden Inhalt an (`AutoSize = true`). Wenn Sie bestimmte Anzeigebereiche rechtsbündig in der Statusleiste ausrichten möchten, fügen Sie vor diesen Anzeigebereichen ein `ToolStripStatusLabel`-Feld ein, dessen `Spring`-Eigenschaft Sie auf `true` setzen. Das Feld nimmt dann unabhängig von der Breite des Fensters den verbleibenden Platz in der Statusleiste ein und drückt die daneben liegenden Felder an den Rand. Es fungiert sozusagen wie eine Stahlfeder (englisch: `spring`).

```
StatusStrip myStatusStrip = new StatusStrip();
myStatusStrip.Parent = this;
myStatusStrip.Dock = DockStyle.Bottom;

ToolStripStatusLabel sbField1 = new ToolStripStatusLabel();
sbField1.Spring = true;
sbField1.TextAlign = ContentAlignment.MiddleLeft;
ToolStripStatusLabel sbField2 = new ToolStripStatusLabel();
sbField2.AutoSize = true;

myStatusStrip.Items.AddRange(new ToolStripItem[] {
    sbField1,
    new ToolStripSeparator(),
    sbField2 });

sbField1.Text = "Feld 1";
sbField2.Text = "Feld 2";
```

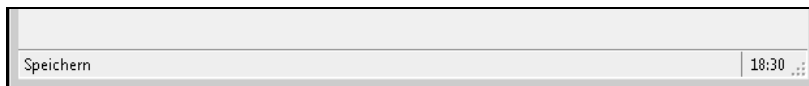


Abbildung 28.21 Die erzeugte Statusleiste

QuickInfo-Texte in Statusleiste einblenden

Lange bevor es die QuickInfo-Texte gab, war es üblich, Hilfetexte zur Funktion der verschiedenen Oberflächenelemente in der Statusleiste anzuzeigen. Der Anwender brauchte dann nur mit der Maus über die diversen Menübefehle, Schaltflächen und sonstigen Steuerelemente zu fahren und konnte in der Statusleiste mitlesen, welche Aktionen mit diesen Elementen verbunden waren. Falls Sie Ihren Anwendern einen ähnlichen Service bieten möchten, gehen Sie wie folgt vor:

1. Behandeln Sie für jedes Steuerelement, zu dem Sie Hinweistexte einblenden möchten, die Ereignisse `MouseHover` und `MouseLeave`.

In der Regel genügt es, für jedes dieser Ereignisse eine einzige Behandlungsmethode zu definieren, die Sie dann mit den verschiedenen Steuerelementen verbinden. Im Code der Methoden können Sie dann ermitteln, welche Art Steuerelement das Ereignis ausgelöst hat, und entsprechend reagieren.

2. Implementieren Sie die Ereignisbehandlungsmethoden. In der Methode(n) für das `MouseHover`-Ereignis kopieren Sie den Hinweistext zu dem auslösenden Objekt in das zugehörige Anzeigefeld der Statusleiste. In der Methode(n) für das `MouseLeave`-Ereignis löschen Sie den Text.

Die nachfolgenden Implementierungen blenden für Menüelemente den Titel und für Symbolschaltflächen den Hinweistext in die Statusleiste ein.

```
private void menuItem_MouseHover(object sender, EventArgs e)
{
    if (sender is ToolStripMenuItem)
    {
        sbToolTipField.Text = ((ToolStripMenuItem)sender).Text;
    }
    if (sender is ToolStripButton)
    {
        sbToolTipField.Text = ((ToolStripButton)sender).ToolTipText;
    }
}

private void menuItem_MouseLeave(object sender, EventArgs e)
{
    sbToolTipField.Text = "";
}
```

HINWEIS Der hier gezeigte Code stammt aus dem Texteditor-Projekt, das Teil der Begleitbeispiele zu diesem Buch ist.

Uhrzeit in Statusleiste einblenden

Mithilfe eines passenden Timers und der Klasse `DateTime` ist es ohne große Mühe möglich, ein Feld der Statusleiste in eine *digitale* Uhr zu verwandeln.

Bei Programmstart richten Sie den Timer ein, dessen `Tick`-Behandlungsmethode in ausreichend kurzen Intervallen aufgerufen werden sollte. Wenn Sie die Uhrzeit ohne Sekunden anzeigen, genügt es vermutlich, die Anzeige alle 10 Sekunden zu aktualisieren. Außerdem sollten Sie die aktuelle Uhrzeit einmal per Hand in das Statusfeld schreiben, damit der Anwender nicht auf das erste Auslösen des Timer-Ereignisses warten muss.

In der Behandlungsmethode zu dem `Tick`-Ereignis wiederholen Sie dann einfach den Code, der die aktuelle Uhrzeit in das Statusfeld schreibt.

```
public partial class Form1 : Form
{
    private Timer clockTimer;

    public Form1()
    {
        ...
    }
}
```

```
// Timer einrichten und starten
clockTimer = new Timer();
clockTimer.Interval = 10000;           // alle 10 sec.
clockTimer.Tick += new EventHandler(this.OnTimer);
clockTimer.Start();

...

// Uhrzeit bei Programmstart von Hand setzen
sbField2.Text = DateTime.Now.ToShortTimeString();
}

private void OnTimer(object sender, EventArgs args)
{
    // Uhrzeit aktualisieren
    sbClockField.Text = DateTime.Now.ToShortTimeString();
}

...
```

HINWEIS Der hier gezeigte Code ist eine leichte Abwandlung des Codes aus dem Texteditor-Projekt, das Teil der Begleitbeispiele zu diesem Buch ist, siehe Vorwort.

Webbrowser

Mithilfe des Webbrowser-Steuerelements `WebBrowser` aus dem Namespace `System.Windows.Forms` können Sie den Inhalt einer Webseite in ein Formular laden.

Fügen Sie dazu das Steuerelement in das Formular ein, weisen Sie den URL der zu ladenden Webseite an die Eigenschaft `Url` zu – fertig!

```
WebBrowser myWebbrowser = new WebBrowser();
myWebbrowser.Parent = this;
myWebbrowser.Dock = DockStyle.Fill;

myWebbrowser.Url = new Uri(@"http://www.cia.gov");
```



Abbildung 28.22 Eine Webseite wird in ein Formular geladen

