7 Implementieren der Menühandler

Die Frage, wie Menüs und Navigationsstrukturen aussehen sollten, kann zu erhitzten Debatten und Meinungsverschiedenheiten führen. Wenn wir zehn Web- und Anwendungsdesigner in ein Zimmer sperren und ihnen denselben Anforderungskatalog vorlegen, bekommen wir wahrscheinlich zehn unterschiedliche Empfehlungen, wie wir Menü- und Navigationsstrukturen implementieren sollen. Ein Streit über die Vor- und Nachteile bestimmter Lösungen würde entbrennen, eine Einigung wäre nicht zu erwarten. Vielleicht denken Sie, dass die Schönheit eben im Auge des Betrachters liegt. Wir reden hier aber nicht vom Aussehen einer Anwendung. Wir reden über Benutzerfreundlichkeit, intuitive Bedienbarkeit und Effizienz beim Entwurf von Menü- und Navigationsstrukturen. In diesem Kapitel stellen wir eine flexible Lösung für diese Probleme vor. Wir bieten Ihnen mehr als nur eine bestimmte Lösung für eine Menüstruktur. Während wir die von uns empfohlenen Menü- und Navigationskomponenten erstellen, vermitteln wir Ihnen so viele Details, dass Sie die Komponenten ganz einfach an Ihren persönlichen Geschmack anpassen können. Viele Unternehmen, auch Microsoft, haben große Mengen Geld und Zeit aufgewendet, um die Benutzerfreundlichkeit von Microsoft Windows-Anwendungen und Webanwendungen zu erforschen. Windows-Anwendungen gibt es schon viel länger als Webanwendungen, und es gibt auf breiter Basis implementierte Standards, etwa die Standardoberfläche seit dem Erscheinen von Microsoft Office 2000. Wir wollen uns nicht auf diese Standards konzentrieren. Lieber zeigen wir Ihnen ein Menübeispiel für Windows Forms am Ende dieses Kapitels, das Ihnen eine Vorstellung davon vermittelt, wie einfach es mit Microsoft Visual Basic .NET und Windows Forms ist, Menüs zu entwerfen. Es gibt fundamentale Unterschiede zwischen den Navigationsmechanismen von Windows- und Webanwendungen. Wegen der wachsenden Bedeutung von Webanwendungen für Internet und Intranet werden wir uns auf Web Forms konzentrieren und Lösungen vorstellen, bei denen wir die Leistungsfähigkeit von Microsoft ASP.NET und Visual Basic .NET kombinieren.

Unser wichtigstes Ziel in diesem Kapitel ist es, Ihnen einfache, wieder verwendbare Menü- und Navigationsobjekte zur Verfügung zu stellen, die so browserunabhängig wie möglich sind

und eine vollständig datengesteuerte Menü- und Navigationsstruktur für Webanwendungen bieten. Wir wollen Ihnen zeigen, wie diese Objekte aufgebaut sind, damit Sie die Objekte ganz nach Wunsch unverändert benutzen oder an Ihre speziellen Anforderungen anpassen können.

Strategien zum Aufbau von Menüs

Bevor wir uns im Einzelnen damit beschäftigen, auf welche Weise wir unsere Menü- und Navigationskomponenten erstellen, wollen wir einen Blick auf zwei Websites werfen, die vorbildliche Menüs und gute Navigationsmöglichkeiten bieten. Die Microsoft-Website *msn.com* bietet gutes Design und ist einfach zu bedienen. Abbildung 7.1 zeigt die Homepage dieser Site. Am oberen Rand hat sie ein Register für die Hauptbereiche, links befinden sich weitere Navigationsleisten für Unterthemen.

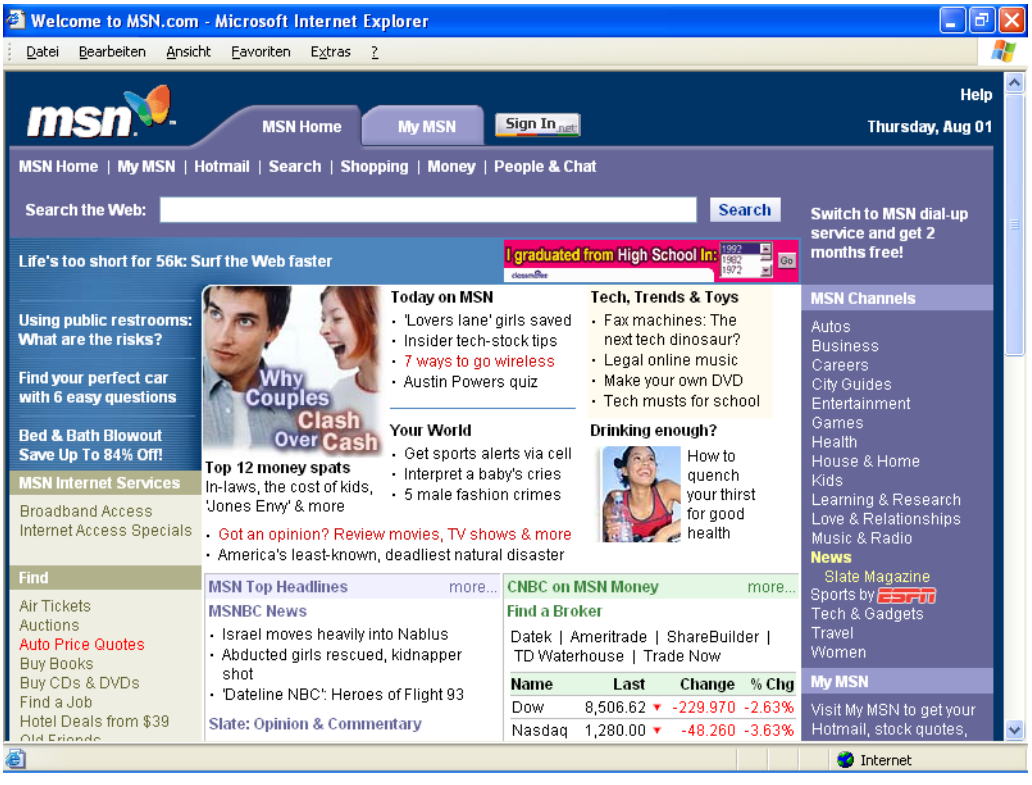

Abbildung 7.1: *Die MSN-Homepage*

Wir kennen viele Websites, die zu komplex sind und bei denen sich der Besucher schwer zurechtfindet. Sogar die fähigsten Benutzer verirren sich auf der Suche nach Informationen. Bestimmt haben Sie ähnliche Erfahrungen gemacht.

Sehen wir uns eine andere Website an, die gute Menü- und Navigationsstrukturen bietet. Abbildung 7.2 zeigt die Website von InfoLink Screening Services, Inc. Diese Site wurde ausschließlich mit .NET entwickelt.

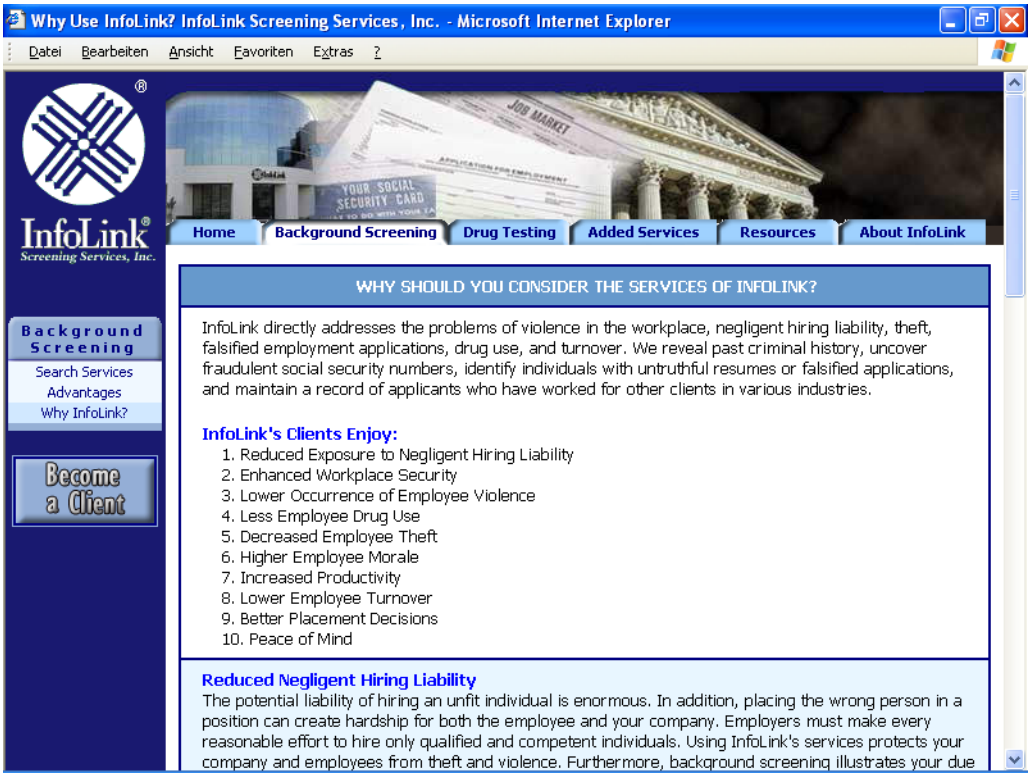

Abbildung 7.2: *Menü- und Navigationsstruktur der Website von InfoLink Screening Services*

Wir hatten die Möglichkeit, diese Webseite zu erstellen. Dank der Erlaubnis der Firma dürfen wir diese Bilder als Beispiele benutzen. Die Register im Hauptmenü geben die Hauptthemenbereiche wieder. Jeder Menüeintrag enthält Unterthemen, die in der Navigationsleiste am linken Rand angezeigt werden. Dieser Ansatz eignet sich besonders für datengesteuerte Menü- und Navigationsstrukturen wie die aus Abbildung 7.2. Abbildung 7.3 zeigt, wie der datengesteuerte Teil der InfoLink-Website aussieht, nachdem sich der Benutzer angemeldet hat. Das Register am oberen Rand enthält wiederum die Hauptthemenbereiche. Die Navigationsleiste am linken Rand bietet Einstiegspunkte für Dateneingabeseiten und Berichte. Weitere Auswahlmöglichkeiten enthält die nummerierte Struktur in der Mitte der Seite. Dieser Ansatz ist einfach und geradlinig. Er macht es den Benutzern leicht, sich auf der Website zurechtzufinden.

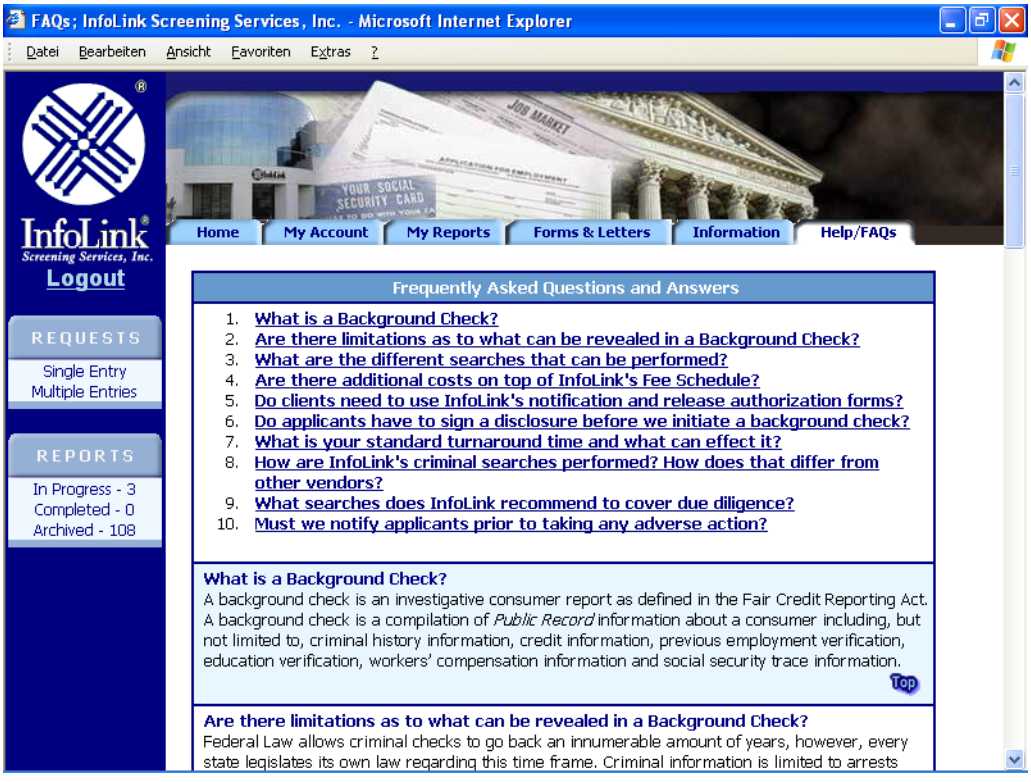

Abbildung 7.3: *Datengesteuerte Menüstruktur*

Unser Ansatz für gute Menüs

Wir können unseren Ansatz gleich zu Anfang kompakt zusammenfassen: »Keep it simple«, so schlicht wie möglich. Unsere Menü- und Navigationsstrukturen sollen folgende Anforderungen erfüllen:

- Sie sollen einfach zu bedienen sein.
- Sie sollen selbsterklärend sein.
- Sie sollen eine relativ flache Struktur aufweisen. Tief verschachtelte Navigationsstrukturen führen die Benutzer schnell in die Irre.
- Sie sollen datengesteuert sein.
- Sie sollen gut aussehen.
- Sie sollen einfach zu implementieren sein.
- Sie sollen .NET-Komponenten nutzen.

Allgemeiner Aufbau der Menüoberfläche

Die Auswertung der weiter oben vorgestellten Menü- und Navigationsstrukturen ergibt den Aufbau aus Abbildung 7.4.

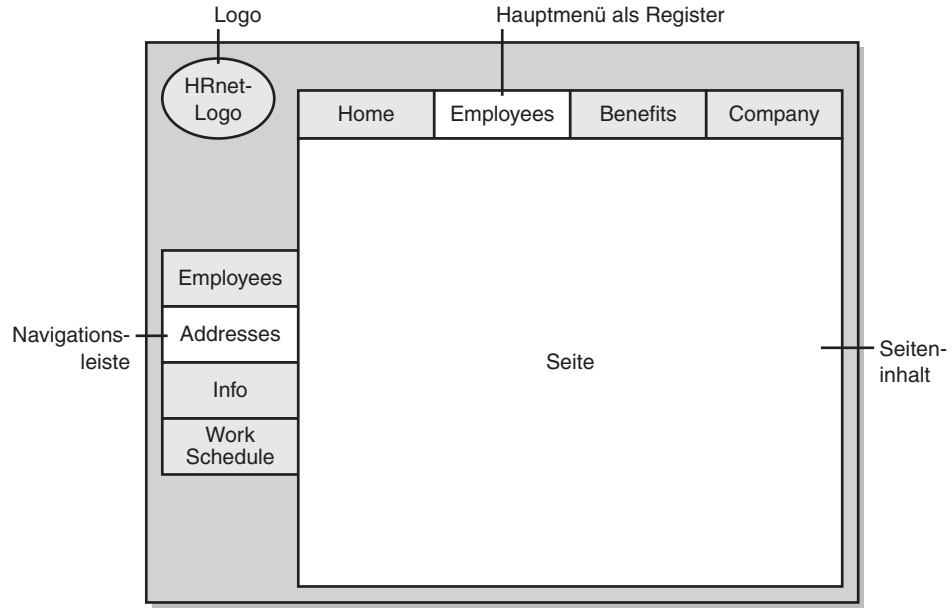

Abbildung 7.4: *Entwurf für die Menü- und Navigationsstruktur von HRnet*

Der Bildschirm ist grafisch und funktional in vier Bereiche unterteilt:

- **Logo** Neben der reinen Darstellung des Logos können wir ihm auch eine Funktion zuweisen, zum Beispiel kann ein Klick auf das Logo zurück zur Homepage führen. Wir könnten auch eine Abmeldefunktion bereitstellen, wie wir sie in Kapitel 10 vorstellen.
- **Hauptmenüregister** Dieser Bereich enthält das zentrale Navigationssteuerelement der Webanwendung. Alle anderen Navigationsleisten sind diesem Registermenü untergeordnet. Wir empfehlen, die Homepage zum ersten Registerelement des Hauptmenüs zu machen.
- **Navigationsleiste** Die Navigationsleiste bildet die zweite Ebene in unserer Menü- und Navigationsstruktur. Wir belassen es bei dieser Ebene, tiefer wollen wir unsere Menüstruktur nicht machen. Wir haben zwar Menübäume gesehen, die mehrere Ebenen tief sind, raten aber von diesem Ansatz ab. Baumstrukturen werden schnell zu breit. Einer Baumstruktur mit Navigationsoptionen zu folgen, wird schnell komplex und unübersichtlich. Der Benutzer übersieht Informationen und ist verwirrt.
- **Seiteninhalt** In diesem Bereich werden die Webinhalte oder die datengesteuerten Informationen angezeigt.

Unser Ansatz mit den vier Bereichen macht es einfacher, wieder verwendbare Seitenvorlagen zu erstellen, bei denen datengesteuerte Navigation und Seitenerstellung angeboten werden. Eine vollständige Implementierung finden Sie in Kapitel 10.

Mehr als zwei Menüebenen zur Verfügung zu stellen, ist nicht ganz einfach. Eine Möglichkeit wäre, im Kernbereich eine Seite anzuzeigen, die ihrerseits ein Register hat. Abbildung 7.5 zeigt, wie das aussehen könnte.

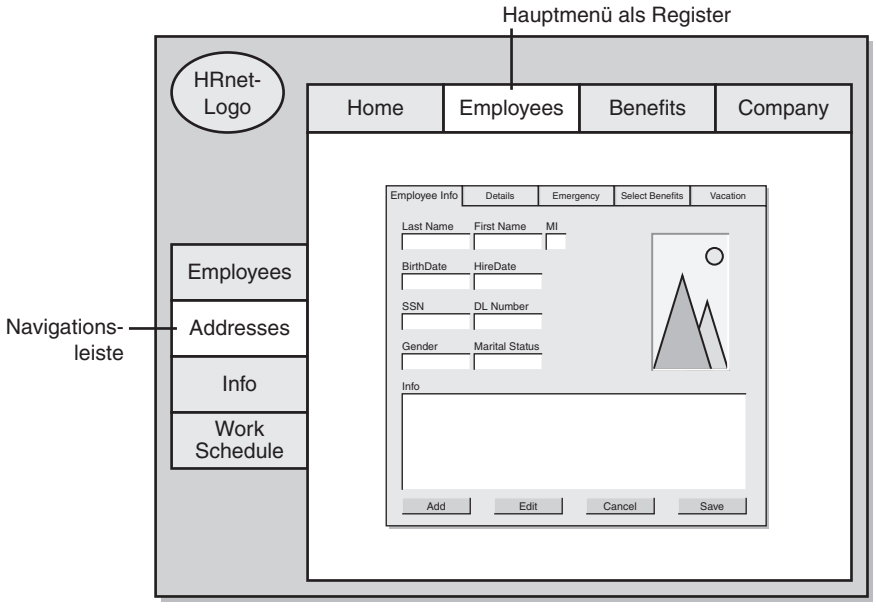

Abbildung 7.5: *Eine weitere Menüebene im Seitenbereich*

Funktionsumfang der Menüs

Die Komponenten, in denen unsere Navigationsstruktur implementiert ist, sollen so leistungsfähig und flexibel wir möglich sein. Unsere Komponenten sollen in der Lage sein, einerseits eigenständig zu funktionieren, andererseits aber auch Verknüpfungen zu den jeweiligen übergeordneten Steuerelementen herzustellen. Die Benutzer müssen das Hauptmenüregister bedienen können, ohne auf andere Menüs zugreifen zu müssen. Wird die Hauptmenüleiste dagegen mit einer seitlich angeordneten Navigationsleiste kombiniert, muss sie steuern können, welche Einträge der seitlichen Navigationsleiste sichtbar und aktivierbar sind. Falls zum Beispiel im Hauptmenü das Register *Homepage* angeklickt wird, verschwindet die seitliche Navigationsleiste; aktiviert der Benutzer das Register *Employees* (Angestellte), wird die Navigationsleiste automatisch sichtbar und zeigt die verfügbaren Punkte an. Natürlich muss dies vollständig datengesteuert geschehen. Der Entwickler braucht lediglich die Komponenten einzusetzen, die Struktur aus Menü und Untermenü zu entwerfen und die Seiteninhalte bereitzustellen. Den Rest erledigen unsere Komponenten. Es sollen mehrere Datenoptionen zur Verfügung stehen:

- **Erstellte Komponente** Während der Initialisierung der Anwendung kann die Menüstruktur an unsere Komponente übergeben werden. Das passiert auf ähnliche Weise wie beim Übergeben von Argumenten an einer gespeicherte Prozedur. Jeder Eintrag im Hauptmenü oder in der Navigationsleiste wird in der Komponente durch eine Zeile dargestellt. Die Einträge legen auch Abhängigkeiten und die Sichtbarkeit fest. Die Komponente erstellt daraus eine Tabellenstruktur, die für die Navigation ausgewertet wird.
- **Datentabellen** Statt die Tabellenstruktur wie im letzten Punkt dynamisch zu erstellen, können vordefinierte Datentabellen übergeben und benutzt werden.
- **XML-Strukturen** Neben den schon erwähnten Lösungen kann eine XML-Struktur übergeben und benutzt werden. Diese Struktur enthält das vollständige Schema und den Inhalt der benötigten Datentabellen.

Wir wollen in der Komponente außerdem Farbe und Größe der Schrift, Hintergrund- und Vordergrundfarbe der Register, Mausverfolgung, QuickInfos und die Größe des Registers einstellen können. Abbildung 7.6 zeigt, welches Ergebnis wir in diesem Kapitel noch erzielen werden.

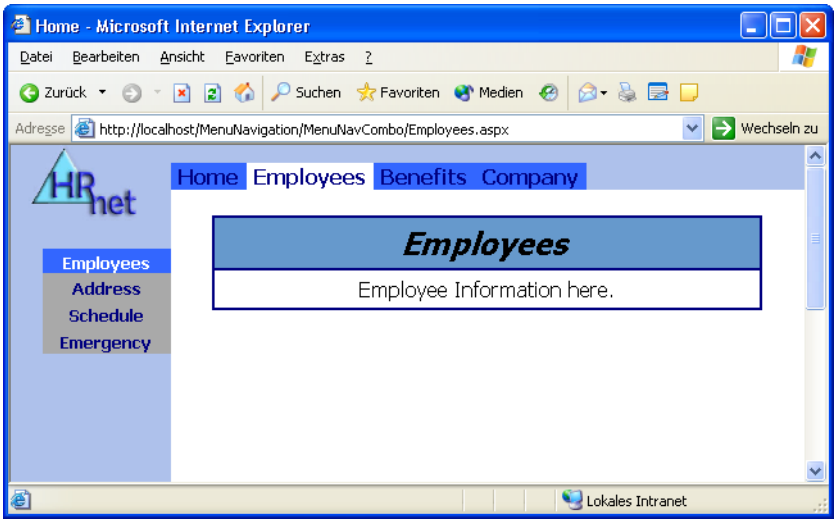

Abbildung 7.6: *Die datengesteuerten Menü- und Navigationsleisten von HRnet*

Erstellen der Objekte für Menü und Navigationsleiste

Während wir die Objekte für Menü und Navigationsleiste von HRnet erstellen, vermitteln wir Ihnen so viele Detailinformationen wie möglich. Wir wollen sicher sein, dass Sie unsere Gedankengänge und Programmiertechniken nachvollziehen können. Nur so werden Sie in der Lage sein, die Struktur und den Funktionsumfang an Ihre eigenen Anwendungen anzupassen.

Die eingesetzte .NET-Technologie

ASP.NET-Serversteuerelemente bieten für unser Menü mehrere Vorteile. Sie generieren browserunabhängigen Code, verfügen über erweiterte Steuerungsmöglichkeiten, die über Deklarationen oder vom Programmcode aus eingestellt werden können, und sie werden auf dem Server gesteuert. Die beiden nächstliegenden Kandidaten für die Navigation sind das `Hyperlink`-Serversteuerelement und das `LinkButton`-Serversteuerelement. Die Schwierigkeit bei der Implementierung dieser Steuerelemente liegt darin, dass die Steuerelementhierarchie auf der Basis von Laufzeitdaten dynamisch erstellt werden muss. Keines der beiden Steuerelemente verfügt über die Eigenschaft `DataSource`. Hier ist das ASP.NET-Serversteuerelement `DataList` die Rettung. Wenn wir `Hyperlink` oder `LinkButton`-Serversteuerelemente in ein anderes Steuerelement einbetten, das die Datenbindung unterstützt, können sie einzeln an die Daten des übergeordneten Steuerelements gebunden werden. Dank dieses kleinen Tricks können wir unsere datengesteuerte Navigation relativ einfach verwirklichen. Unser erstes Beispiel ist *SimpleLinkButton.aspx* aus dem Unterverzeichnis *Ch07/MenuNavigation/LinkButton*. Es ist eine einfache Implementierung dieser Strategie. Da

diese Strategie die Grundlage für alle unsere Navigationssteuerelemente bildet, wollen wir uns zuerst damit beschäftigen.

Sehen wir uns erst einmal die Code-Behind-Datei an. Wenn die Seite zum ersten Mal dargestellt wird, erstellen wir eine Datentabelle mit den benötigten Feldern und binden sie an das `DataList`-Steuerelement. Wir wollen Ihnen anhand dieser Tabelle zeigen, wie Sie Daten indirekt an das `LinkButton`-Steuerelement binden. Der folgende Code erstellt die Tabelle und fügt die erste Zeile hinzu:

```
Private Sub Page_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
    ' Code zum Initialisieren der Seite hier einfügen.
    If Not IsPostBack Then
        ' Die MainMenuTable erstellen und mit Beispiele füllen.
        Dim menuTable As New DataTable("MainMenuTable")
        Dim menuColumn As DataColumn
        Dim row As DataRow
        menuColumn = menuTable.Columns.Add("MenuID", System.Type.GetType("System.Int32"))
        menuColumn = menuTable.Columns.Add("MenuTabName", System.Type.GetType("System.String"))
        menuColumn = menuTable.Columns.Add("MenuTabTip", System.Type.GetType("System.String"))
        menuColumn = menuTable.Columns.Add("MenuTabURL", System.Type.GetType("System.String"))
        menuColumn = menuTable.Columns.Add("HasNavBar", System.Type.GetType("System.Boolean"))
        ' Erster Eintrag: Register für die Homepage
        row = menuTable.NewRow
        row("MenuID") = 1
        row("MenuTabName") = "Home"
        row("MenuTabTip") = "Our Home Page"
        row("MenuTabURL") = "Home.aspx"
        row("HasNavBar") = False
        menuTable.Rows.Add(row)
        ⋮
        MainMenuList.DataSource = menuTable
        MainMenuList.DataBind()
```

Diese Tabellenstruktur enthält alle Daten, die für ein Registersteuerelement nötig sind. Das sind Name, QuickInfo-Daten, Ziel-URL und sogar ein `Boolean`-Wert, der festlegt, ob das Element mit einer Navigationsleiste verknüpft ist. Diesen `Boolean`-Wert werden wir in späteren Beispielen benutzen.

Sehen wir uns den HTML-Code zu diesem Beispiel an. `DataList`-Steuerelemente können über Eigenschaftenseiten gesteuert werden, wir raten aber vorerst davon ab. Sie werden die interne Funktionsweise des `DataList`-Steuerelements besser verstehen, wenn Sie wissen, was im HTML-Code passiert.

```
<asp:DataList id="MainMenuList" RepeatDirection="Horizontal"
    runat="server">
    <ItemTemplate>
        <asp:LinkButton ID="idMenuLink"
            ToolTip='<%# Container.dataitem("MenuTabTip")%>'
            Text='<%# Container.dataitem("MenuTabName")%>'
            CommandArgument='<%# Container.dataitem("MenuTabURL") %>'
            Runat="Server">
        </asp:LinkButton>
    </ItemTemplate>
</asp:DataList>
```

Einige Punkte an diesem HTML-Code sind bemerkenswert. Das `DataList`-Steuerelement wird in der Standardeinstellung vertikal angeordnet. Daher müssen wir dem Attribut `RepeatDirection`

im Element asp:DataList den Wert Horizontal zuweisen. Anschließend brauchen wir eine Vorlage innerhalb der DataList, mit der wir ein LinkButton-Steuerelement einbetten können. In der Deklaration des LinkButton-Steuerelements können wir die Datenbindung veranlassen, die das LinkButton-Steuerelement mit dem DataList-Steuerelement verknüpfen. Der folgende Codeausschnitt, den Sie bereits aus dem letzten Ausschnitt kennen, zeigt die Befehle für diese Datenbindung in Fettschrift.

```
<asp:LinkButton ID="idMenuLink"
    ToolTip='<%# Container.dataitem("MenuTabTip")%>'
    Text='<%# Container.dataitem("MenuTabName")%>'
    CommandArgument='<%# Container.dataitem("MenuTabURL") %>'
    Runat="Server">
</asp:LinkButton>
```

Wir binden das dataitem des Container-Objekts an bestimmte Teile des LinkButton-Steuerelements. Im Fall des QuickInfos (ToolTip) handelt es sich um das Feld MenuTabTip der menuTable, die mit dem DataList-Steuerelement MainMenuList verknüpft ist. Das Attribut Text bestimmt die Beschriftung des Menüregisters. Das CommandArgument des LinkButton-Steuerelements macht es möglich, dass wir serverseitig ein Click-Ereignis des LinkButton auslösen und ihm den gewünschten URL übergeben können. Der Inhalt wird nicht auf der Seite angezeigt, sondern im ItemCommand-Ereignis übergeben. Das wird in der Code-Behind-Datei des Beispiels erledigt:

```
Private Sub MainMenuListItemCommand(ByVal source As Object, _
    ByVal e As System.Web.UI.WebControls.DataListCommandEventArgs) _
    Handles MainMenuList.ItemCommand
    MainMenuList.SelectedIndex = e.Item.ItemIndex
    lblMessage.Text = "Menu Index = " & e.Item.ItemIndex
    lblMessage.Text += " URL Target = " & e.CommandArgument.ToString
    ' Response.Redirect(e.CommandArgument.ToString)
End Sub
```

Die wichtigste Fähigkeit, die DataList uns in der Kombination mit dem LinkButton-Steuerelement zur Verfügung stellt, ist das Ereignis ItemCommand. Bei der Unterroutine, die das Click-Ereignis unseres DataList-Steuerelements verarbeitet, definieren wir die DataListCommandEventArgs unseres MainMenuList.ItemCommand-Objekts als Parameter.

```
Private Sub MainMenuListItemCommand(ByVal source As Object, _
    ByVal e As System.Web.UI.WebControls.DataListCommandEventArgs) _
    Handles MainMenuList.ItemCommand
    ⋮
```

Dieser Ereignishandler stellt uns zwei der wichtigsten Parameter bereit, die wir benötigen. Erstens erhalten wir den ItemIndex des DataList-Steuerelements, das angeklickt wurde. So können wir später den Zustand des Menüs an die Anwendung übergeben. Zweitens erhalten wir das CommandArgument, das dem DataList-Steuerelement zugeordnet ist. In unserem Fall handelt es sich um den URL der Seite, auf die wir den Browser umleiten. Damit wir die Seite *SimpleLinkButton.aspx* einfacher testen können, leiten wir den Browser nicht um, sondern schreiben die Ausgaben in das Beschriftungsfeld lblMessage.

```
lblMessage.Text = "Menu Index = " & e.Item.ItemIndex
lblMessage.Text += " URL Target = " & e.CommandArgument.ToString
```

Ein weiterer interessanter Punkt ist die Eigenschaft SelectedIndex des DataList-Steuerelements. Damit können wir festlegen, welches Element der DataList momentan ausgewählt ist. Wir haben die folgende Zeile in das vorhergehende Beispiel eingefügt, damit das DataList-Element ausgewählt bleibt, das angeklickt wurde:

```
MainMenuList.SelectedIndex = e.Item.ItemIndex
```

Jetzt ist es an der Zeit, unsere erste Menüseite auszuprobieren. Sie sieht zugegebenermaßen nicht besonders aufregend aus, bietet aber den gewünschten Funktionsumfang:

- Sie ist vollständig datengesteuert und erstellt sich selbst dynamisch.
- Sie verwendet .NET-Komponenten.
- Sie verarbeitet selbst ihren Zustand.
- Sie erlaubt die Umleitung auf andere Seiten.

Ergänzen wir dieses Beispiel nun durch das Aussehen und den Funktionsumfang, die uns vorschweben. Unser zweites Beispiel, *BetterLinkButton.aspx*, ist das Ergebnis der Optimierung unseres `DataList`-Steuerelements. Die Änderungen sind rein optisch, die Funktionsweise des Steuerelements hat sich in keiner Weise verändert. Wir nutzen Style Sheets und bereits vorhandene Optionen im `DataList`-Steuerelement. Der folgende Ausschnitt zeigt den Style Sheet, den wir mit der Datei verknüpft haben. Er steuert das Aussehen des HTML-Bodys und des `LinkButton`-Steuerelements.

```
<style>
    A.menutext { FONT-WEIGHT: bold; FONT-SIZE: 12pt; MARGIN-LEFT: 5px;
        COLOR: navy; MARGIN-RIGHT: 5px; TEXT-DECORATION: none }
    A.menutext:hover { COLOR: blue; TEXT-DECORATION: none }
    A.menutext:visited { COLOR: navy; TEXT-DECORATION: none }
    A.menutext:hover { COLOR: blue; TEXT-DECORATION: none }
    A.menutext:visited:hover {text-decoration: none; color: blue}
    BODY { FONT-WEIGHT: normal; FONT-SIZE: 10pt; MARGIN: 0px;
        WORD-SPACING: normal; TEXT-TRANSFORM: none; FONT-FAMILY: Tahoma,
        Arial, Helvetica, Sans-Serif; LETTER-SPACING: normal;
        BACKGROUND-COLOR: #aec1eb }
</style>
```

Der HTML-Body übernimmt die Designänderungen automatisch, der `DataList` müssen wir allerdings mitteilen, welchen Stil sie verwenden soll:

```
<asp:LinkButton ID="idMenuLink" cssclass="menutext" ...
```

Der Abschnitt `menutext` aus der Style Sheet-Datei weist dem `LinkButton`-Steuerelement Schriftart, Schriftfarbe und Aktivierungsfarbe zu. Beachten Sie, dass wir die Unterstreichung entfernt haben und dafür sorgen, dass bereits besuchte Links genauso aussehen wie normale Links. Wir wollen nicht das Verhalten eines Links nachnahmen, bei dem unterschiedliche Farben für bereits besuchte und neue Ziele verwendet werden und die durch Unterstreichung hervorgehoben werden. Außerdem haben wir den Links eine etwas andere Farbe zugewiesen, wenn sich der Mauszeiger darüber befindet. Experimentieren Sie mit diesen Einstellungen, damit Sie ein Gefühl dafür bekommen, auf welche Weise Sie das Aussehen des `LinkButton`-Steuerelements beeinflussen können.

Als Nächstes wollen wir die Hintergrundfarbe des angeklickten Elements innerhalb des `DataList`-Steuerelements ändern. Auf diese Weise erhält die `DataList` das Aussehen eines Registersteuerelements. Erfreulicherweise verfügt `DataList` bereits über Fähigkeiten, die das ganz einfach machen. Wir erstellen zwei Vorlagen innerhalb der `DataList`, sie heißen `ItemStyle` und `SelectedItemStyle`:

```
<SelectedItemStyle BackColor="White"></SelectedItemStyle>
<ItemStyle BackColor="#3366FF"></ItemStyle>
```

Experimentieren Sie auch mit diesem Code. Sie können die Standardeinstellungen der Werte jederzeit wiederherstellen. Neben `BackColor` gibt es viele weitere Eigenschaften, mit denen sich das Verhalten von ausgewählten oder nicht ausgewählten Elementen verändern lässt. Weiter

unten in diesem Kapitel zeigen wir Ihnen, wie Sie diese Eigenschaft zur Laufzeit einstellen statt wie in den bisherigen Beispielen über die Deklaration.

Wir verfügen jetzt über die Grundlagen, mit denen wir unseren restlichen Menücode angehen können. Bevor wir das Beispiel in ein wieder verwendbares Benutzersteuerelement verwandeln, möchten wir erklären, warum wir statt des `Hyperlink`-Serversteuerelements das `LinkButton`-Serversteuerelement verwenden. Wir haben die Erfahrung gemacht, dass sich das `Hyperlink`-Serversteuerelement genau wie ein normaler HTML-Hyperlink verhält. Es leitet den Benutzer innerhalb des Browsers von einer Webseite zur anderen. Das macht es uns unmöglich, einen Roundtrip zum Server zu erzwingen. Das `LinkButton`-Steuerelement ermöglicht einen Server-Roundtrip und kann zusätzlichen Code ausführen, bevor es eine serverseitige Umleitung zu einer anderen Seite auslöst. Das ist wichtig für uns, weil wir auf diese Weise die Zustandsdaten innerhalb des Steuerelements einstellen können. In diesem Fall speichern die Zustandsdaten den Index der angeklickten Schaltfläche, so dass wir sie auf der Zielseite automatisch auswählen können. Zusätzlich können wir später mit Hilfe dieser Zustandsdaten die Informationen für die Menütabelle auslesen. Wir können die Zustandsdaten von Hauptmenü und Navigationsleiste entweder getrennt oder zusammen auswerten. So bekommen wir immer korrekte Zustandsdaten zurück, egal ob wir das Hauptmenü allein oder in Kombination mit der Navigationsleiste benutzen. Würden wir diese Zustandsdaten vor dem Seitenwechsel nicht speichern, müssten wir uns eine Methode einfallen lassen, wie sich jede Seite aus unserer Webanwendung selbst bei den Menüsteuerelementen anmeldet. Das wäre zu aufwendig. Wir können die Menü- und Navigationsstruktur nur dann in unseren Komponenten kapseln, wenn die Komponente ihre Zustandsdaten selbst verwalten kann.

Entwerfen des Steuerelements

Nach einem Blick auf das letzte Beispiel könnten wir der Versuchung erliegen, loszustürmen und sofort ein Benutzersteuerelement oder ein benutzerdefiniertes Serversteuerelement zu erstellen. Rufen wir uns aber noch einmal die Ratschläge aus Kapitel 2 ins Gedächtnis und planen wir das Steuerelement erst.

Wir beginnen mit den benötigten Datentabellen und ihrer Struktur. Beide weiter oben gezeigten Beispiele enthalten bereits eine Datentabelle für das Hauptmenü. Wir können ihre Struktur als Ausgangspunkt nehmen und verbessern. Tabelle 7.1 zeigt die Struktur der Tabelle *MainMenuTable*.

Feldname	Feldtyp	Nullwerte erlaubt	Index
MenuID	`Integer`	Nein	Primärschlüssel
MenuTabName	`String`	Nein	
MenuTabTip	`String`	Ja	
MenuTabURL	`String`	Nein	
HasChild	`Boolean`	Nein; Standardeinstellung ist `False`	

Tabelle 7.1: *MainMenuTable*

Wir haben einen Primärschlüssel mit dem Feldnamen *MenuID* hinzugefügt und benötigte Felder definiert. Damit die Tabelle *MainMenuTable* steuern kann, ob ihre untergeordneten Elemente sichtbar sind, erstellen wir eine 1:n-Beziehung zwischen ihr und der Tabelle *NavBarTable*. Tabelle 7.2 zeigt ihre Struktur.

Feldname	Feldtyp	Null erlaubt	Index
NavBarID	`Integer`	Nein	Primärschlüssel
MenuID	`Integer`	Nein	Fremdschlüssel
NavBarName	`String`	Nein	
NavBarTip	`String`	Ja	
NavBarURL	`String`	Nein	
HasChild	`Boolean`	Nein; Standardeinstellung ist `False`	

Tabelle 7.2: *NavBarTable*

Beide Tabellen werden in einer Komponente namens `MenuData` angelegt, die wir in Kürze hinzufügen. In dieser Komponente erstellen wir auch die Beziehung zwischen den Tabellen. Weitere Ebenen lassen sich hinzufügen, indem weitere Tabellen mit derselben Struktur wie *NavBarTable* definiert und jeweils in einer 1:n-Beziehung verknüpft werden. Vergessen Sie aber nicht, dass wir eine Navigationsstruktur empfehlen, die nur zwei oder drei Ebenen tief ist.

Nachdem die Komponente `MenuData` fertig ist, verschieben wir unser `MainMenu` in ein Benutzersteuerelement und erweitern es mit der Fähigkeit der Zustandsverwaltung. Dieses Steuerelement packen wir in eine Vorlage, die wir für die Seiten *Home* (Homepage), *Employee* (Angestellte), *Benefits* (Lohnzusatzleistungen) und *Company* (Unternehmen) verwenden. Aus diesen Seiten erstellen wir eine kleine menügesteuerte Anwendung.

Nachdem wir das Beispiel getestet haben, fügen wir zu der Komponente `MenuData` und dem Benutzersteuerelement weitere Fähigkeiten hinzu, zum Beispiel Optionen für Farben, Schriftart, Größe, Hintergrund im ausgewählten oder nicht ausgewählten Zustand.

Wenn das Registersteuerelement für das Hauptmenü fertig ist, können wir diese Schritte wiederholen, um eine vertikale Navigationsleiste zu erstellen. Sie werden bald sehen, dass die Navigationsleiste auf praktisch dieselbe Weise entwickelt wird wie das Hauptmenü. Wir müssen aber weitere Optionen für die Verwaltung von Zustand und Menüdaten hinzufügen. Machen wir uns also ans Werk.

Die Komponente *MenuData*

Sie finden unsere Komponente unter dem Namen *MenuData* im Unterverzeichnis *Ch07\MenuNavigation\MenuData*. Ein Teil ihres Funktionsumfangs gleicht dem unserer Datenzugriffskomponente aus Kapitel 4. Wir versuchen immer, Ereignisprotokollierung zu implementieren und das Löschverhalten zu ändern. Bei diesem Beispiel haben wir keine öffentlichen Konstruktoren implementiert, weil wir nur eine Möglichkeit anbieten, die Komponente zu initialisieren und zu benutzen. Die Komponente `MenuData` erstellt ein `DataSet`, das die Steuerungsdaten für das Menü und die Navigationsleiste enthält. Diese Steuerungsdaten werden von den aufrufenden Methoden erstellt. Dies ähnelt der Weise, in der bei unserer Datenzugriffskomponente Parameter hinzugefügt werden. Jedes Mal, wenn eine Methode aufgerufen wird, erstellt diese Methode einen weiteren Eintrag in einer der Menüdatentabellen.

Beim Initialisieren der Komponente `MenuData` legen wir zuerst die Struktur unserer Tabellen an. Dazu dient der folgende Code aus dem Codeabschnitt *Vom Component Designer generierter Code*:

```
Public Sub New()
    MyBase.New()
    ' Dieser Aufruf ist für den Komponenten-Designer erforderlich.
    InitializeComponent()
```

```
    ' Initialisierungen nach dem Aufruf InitializeComponent() hinzufügen
    ' Struktur des DataSet anlegen
    privateDataSet = CreateMenuDataSet()
End Sub
```

Die fett hervorgehobene Zeile haben wir eingefügt, um die Funktion `CreateMenuDataSet` aufzurufen. Diese Funktion weist `privateDataSet` aus unserer Komponente die Struktur unserer Tabellen zu. Im Objekt `privateDataSet` speichern wir die Einträge für unsere Menü- und Navigationsstruktur. Schließlich geben wir die Einträge aus unserer Komponente zurück.

Der folgende Code demonstriert, wie wir die Tabellen und ihre Beziehungen anlegen:

```
Function CreateMenuDataSet() As DataSet
    Dim privateMenuDataSet As New DataSet("MenuTables")
    Dim privateMenuTable As New DataTable("MainMenuTable")
    ' Die DataTable des Menüs definieren.
    Dim privateMenuColumn As DataColumn
    Try
        privateMenuColumn = privateMenuTable.Columns.Add("MenuID", _
            System.Type.GetType("System.Int32"))
        privateMenuColumn.AllowDBNull = False
        privateMenuColumn = privateMenuTable.Columns.Add( _
            "MenuTabName", System.Type.GetType("System.String"))
        privateMenuColumn.AllowDBNull = False
        ⋮
    Catch tableMenuException As Exception
        Throw New Exception(privateExceptionMessage & _
            " Main Menu Table creation error.", tableMenuException)
    End Try
    Dim privateNavTable As New DataTable("NavTable")
    Dim privateNavColumn As DataColumn
    Try
        privateNavColumn = privateNavTable.Columns.Add("MenuID", _
            System.Type.GetType("System.Int32"))
        privateNavColumn.AllowDBNull = False
        privateNavColumn = privateNavTable.Columns.Add("NavBarId", _
            System.Type.GetType("System.Int32"))
        privateNavColumn.AllowDBNull = False
        ⋮
    Catch tableNavException As Exception
        Throw New Exception(privateExceptionMessage & _
            " NavBar Table creation error.", tableNavException)
    End Try
    privateMenuDataSet.Tables.Add(privateMenuTable)
    privateMenuDataSet.Tables.Add(privateNavTable)
    Try
        privateMenuDataSet.Relations.Add("MenuNav",
            privateMenuDataSet.Tables("MainMenuTable").Columns("MenuID"), _
            privateMenuDataSet.Tables("NavTable").Columns("MenuID"))
        Return privateMenuDataSet
    Catch linkException As Exception
        Throw New Exception(privateExceptionMessage & _
            " One-to-many Link could not be created.", linkException)
    End Try
End Function
```

In den ersten Zeilen erstellen wir das `DataSet` und die `DataTable`-Objekte mit ihren jeweiligen Feldern. Wir definieren für alle Felder den Datentyp und legen fest, welche Felder einen Wert

haben müssen. Nach dem Anlegen der Felder fügen wir sie zu dem `DataSet` hinzu und legen eine Beziehung an. Die Struktur in diesem Beispiel legt die Tabellen für ein Hauptmenü und eine untergeordnete Navigationsleiste an. Falls Sie weitere Ebenen in Ihrem Menü benötigen, brauchen Sie lediglich für jede Ebene eine zusätzliche Tabelle zu erstellen, die mit der *NavBarTable* verknüpft ist.

Im nächsten Abschnitt unserer Komponente `MenuData` erstellen wir Zeile für Zeile die Menütabellen. Sehen wir uns diesen Code genauer an. Es handelt sich um den Codeabschnitt *AddMainMenuItems & AddNavBarItems*:

```
Public Sub AddMainMenuParameter(ByVal MenuID As Integer, _
    ByVal MenuTabName As String, _
    ByVal MenuTabURL As String, _
    ByVal HasChild As Boolean, _
    Optional ByVal MenuTabTip As String = Nothing)
    Dim usedMenuTable As DataTable   ' Die Menütabelle holen.
    Dim addRow As DataRow
    Try
        usedMenuTable = privateDataSet.Tables("MainMenuTable")
        addRow = usedMenuTable.NewRow
        addRow("MenuID") = MenuID
        addRow("MenuTabName") = MenuTabName
        addRow("MenuTabURL") = MenuTabURL
        addRow("HasChild") = HasChild
        addRow("MenuTabTip") = MenuTabTip
        usedMenuTable.Rows.Add(addRow)
    Catch addMainMenuException As Exception
        ' Ausnahme mit der privaten Funktion LogException protokollieren.
        LogException(addMainMenuException)
        ' Ausnahme wird an den Aufrufer weitergegeben.
        Throw New Exception(privateExceptionMessage & _
            " Adding a record to the Main Menu Table failed.", addMainMenuException)
    End Try
End Sub
```

Wir fügen jeweils ein Element zu dem Hauptmenü oder der Navigationsleiste hinzu, indem wir eine öffentliche Unterroutine aufrufen und dabei Werte für alle Tabellenfelder übergeben. Wir müssen lediglich aufpassen, dass wir die richtigen Datentypen verwenden und optionale Argumente am Ende übergeben. (Vergessen Sie nicht, den optionalen Parametern einen Standardwert zuzuweisen. In diesem Beispiel erhält der QuickInfo-Eintrag den Wert `Nothing`.)

```
Public Sub AddMainMenuParameter(ByVal MenuID As Integer, _
    ByVal MenuTabName As String, _
    ByVal MenuTabURL As String, _
    ByVal HasChild As Boolean, _
    Optional ByVal MenuTabTip As String = Nothing)
```

Wenn wir die Unterroutine aufrufen, ermittelt sie die richtige Menüdatentabelle und fügt eine neue Zeile hinzu. Dann trägt sie die übergebenen Argumente in diese Zeile ein und speichert sie in der `DataTable MainMenuTable`:

```
usedMenuTable = privateDataSet.Tables("MainMenuTable")
addRow = usedMenuTable.NewRow
addRow("MenuID") = MenuID
addRow("MenuTabName") = MenuTabName
addRow("MenuTabURL") = MenuTabURL
addRow("HasChild") = HasChild
addRow("MenuTabTip") = MenuTabTip
usedMenuTable.Rows.Add(addRow)
```

Auf diese Weise können wir Zeile für Zeile zu den Menütabellen hinzufügen. Die Unterroutine `AddMainMenuParameter` trägt Zeilen in die Datentabelle *MainMenuTable* ein, die Unterroutine `AddNavBarParameter` in die Datentabelle *NavBarTable*. Der einzige Unterschied liegt in den Parametern und den Datenfeldern.

Unsere Komponente `MenuData` ist jetzt beinahe bereit für ihren ersten Test, auch wenn noch einige Fähigkeiten fehlen. Im Abschnitt »Das Benutzersteuerelement für die Navigationsleiste« weiter unten in diesem Kapitel werden wir sie erweitern, indem wir die Auswahl von Farben und Schriftart ermöglichen. Die Datei *MenuDataTestForm.aspx* aus dem Verzeichnis *LinkButton* ruft die Komponente `MenuData` auf, fügt zu beiden Tabellen Einträge hinzu und zeigt beide Tabellen in `DataGrid`-Steuerelementen an. Auf diese Weise können wir prüfen, ob die Daten korrekt sind. Wir können auch die Funktionsfähigkeit der Komponente prüfen. Sehen wir uns schnell an, wie diese Komponente benutzt wird. Zuerst importieren wir den Namespace `MenuData`, er enthält die Klasse `MenuDataServer`:

```
Imports MenuNavigation.MenuData
```

Im Ereignishandler `Page_Load` der Seite rufen wir die Komponente `MenuData` auf und fügen die Menüeinträge hinzu. Der gesamte Code des Ereignishandlers sieht so aus:

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    ' Hier Benutzercode zur Seiteninitialisierung einfügen.
    Dim menuTable As DataTable         ' Tabelle für Hauptmenü deklarieren.
    Dim navTable As DataTable          ' Tabelle für Navigationsleiste deklarieren.
    ' Die Komponente MenuTable aufrufen und Daten zu der Tabelle für das Hauptmenü hinzufügen.
    Dim localMenuTables As New MenuDataServer()
    localMenuTables.AddMainMenuParameter(1, "Home", "Home.aspx", False,"Our Home Page")
    localMenuTables.AddMainMenuParameter(2, "Employees", "Employees.aspx", True, "All about Employees")
    localMenuTables.AddMainMenuParameter(3, "Benefits", "Benefits.aspx", True, "Our Company's Benefits")
    localMenuTables.AddMainMenuParameter(4, "Company", "Company.aspx", False)
    ' Die menuTable aus dem DataSet holen.
    menuTable = localMenuTables.GetMenuDataSet.Tables("MainMenuTable")
    ' Einträge in die Tabelle NavBar einfügen.
    localMenuTables.AddNavBarParameter(2, 1, "2nd Tab whatever", "whatever.aspx", False)
    localMenuTables.AddNavBarParameter(2, 2, "2nd Tab whatever2", "whatever2.aspx", False)
    localMenuTables.AddNavBarParameter(3, 1, "3rd Tab whatever", "whatever3.aspx", False)
    navTable = localMenuTables.GetMenuDataSet.Tables("NavTable")
    localMenuTables.Dispose()          ' Methode Dispose des Objekts aufrufen.
    localMenuTables = Nothing          ' Sicherstellen, dass es nicht noch einmal benutzt wird.
    ' Wir weisen die zurückgegebene Tabelle unserer DataList für das Hauptmenü zu.
    MainMenuGrid.DataSource = menuTable
    MainMenuGrid.DataBind()
    ' Wir weisen die zurückgegebene Tabelle unserer DataList für die Navigationsleiste zu.
    NavBarGrid.DataSource = navTable
    NavBarGrid.DataBind()
End Sub
```

Außerdem müssen wir eine Variable für die Klasse `MenuDataServer` anlegen:

```
Dim localMenuTables As New MenuDataServer()
```

Danach können wir die Menüeinträge hinzufügen:

```
localMenuTables.AddMainMenuParameter(1, "Home", "Home.aspx", False,"Our Home Page")
localMenuTables.AddMainMenuParameter(2, "Employees", "Employees.aspx", True, "All about Employees")
⋮
```

Nachdem wir alle Menüeinträge haben, weisen wir das Ergebnis lokal definierten Tabellen zu (das heißt, `DataTable`-Objekten, die in der Code-Behind-Datei der Seite mit dem Menü deklariert wurden). Dazu rufen wir die Funktion `GetMenuData` des `MenuData`-Objekts auf. So erhalten wir ein

DataSet, das alle erstellten Tabellen enthält. Aus diesem Grund müssen wir genau angeben, welche Tabelle aus dem DataSet wir haben wollen:

```
menuTable = localMenuTables.GetMenuDataSet.Tables("MainMenuTable") ' Die menuTable aus dem DataSet holen.
navTable = localMenuTables.GetMenuDataSet.Tables("NavTable")
```

Zuletzt weisen wir dem DataGrid-Steuerelement die lokalen Tabellen zu und aktivieren die Datenbindung:

```
MainMenuGrid.DataSource = menuTable
MainMenuGrid.DataBind()
```

In unserem nächsten Beispiel, *DataLinkButton.aspx*, erstellen wir auf dieselbe Weise eine vollständig datengesteuerte Navigationsleiste. Statt auf andere Seiten zu springen, zeigen wir den Index des ausgewählten Menüeintrags und die Zielseite an, zu der wir gewechselt hätten. Probieren Sie dieses Beispiel bitte aus und verändern Sie den Code etwas. Sie werden sehen, dass wir genau steuern können, was das Menü anzeigt und was nicht. Wir können sogar dafür sorgen, dass während des Page_Load-Ereignisses einer Seite zwischen mehreren Menüregistern gewählt wird. Welches Menü dann geladen wird, könnte zum Beispiel aufgrund der Rolle oder der Berechtigungen des Benutzers entschieden werden.

Das Benutzersteuerelement für das Hauptmenü

Statt den HTML-Code und die Code-Behind-Datei des Menüs in jede einzelne Seite unserer Webanwendung zu kopieren, erstellen wir dieses Steuerelement jetzt in Form eines ASP.NET-Benutzersteuerelements. Wir probieren das erst mal in einer einfachen Version aus, bevor wir die Eigenständigkeit und Kapselung schrittweise steigern.

Die Beispiele aus den nächsten Abschnitten finden Sie im Unterverzeichnis *MenuTabs*. Wir setzen voraus, dass Sie wissen, wie Sie Benutzersteuerelemente erstellen und einsetzen. Die Datei *MainMenuTest.ascx* enthält das Benutzersteuerelement für das Register des Hauptmenüs. Wenn Sie sich den HTML-Code ansehen, werden Sie feststellen, dass er dem aus unserem letzten Beispiel genau gleicht. Die DataList- und LinkButton-Steuerelemente mit ihren jeweiligen Datenbindungen werden auf genau dieselbe Weise benutzt. Die Unterschiede liegen in der Code-Behind-Datei.

Wir wollen die Tabellen für das Hauptmenü und die Navigationsleiste nicht jedes Mal neu erstellen, wenn wir eine Seite aktualisieren oder zu einer anderen Seite gehen. Aus diesem Grund erstellen wir eine Sitzungsvariable für die Menütabellen. Jedes Mal, wenn das Benutzersteuerelement MainMenuTest aufgerufen wird, sucht es nach dieser Sitzungsvariable. Ist sie nicht vorhanden, wird sie angelegt, ansonsten wird sie verwendet. Der Code sieht folgendermaßen aus:

```
If Session("MainMenuTable") Is Nothing Then
    ' Die Komponente MenuTable aufrufen und Einträge in die Tabellen für das Hauptmenü einfügen.
    Dim localMenuTables As New MenuDataServer()
    localMenuTables.AddMainMenuParameter(1, "Home", _
        "/MenuNavigation/MenuTabs/Home.aspx", False, "Our Home Page")
    localMenuTables.AddMainMenuParameter(2, "Employees", _
        "/MenuNavigation/MenuTabs/Employees.aspx", True, _
        "All about Employees")
    localMenuTables.AddMainMenuParameter(3, "Benefits", _
        "/MenuNavigation/MenuTabs/Benefits.aspx", True, _
        "Our Company's Benefits")
    localMenuTables.AddMainMenuParameter(4, "Company", _
        "/MenuNavigation/MenuTabs/Company.aspx", False)
    menuTable = localMenuTables.GetMenuDataSet.Tables("MainMenuTable")
    Session("MainMenuTable") = menuTable
```

```
    localMenuTables.Dispose()          ' Methode Dispose des Objekts aufrufen.
    localMenuTables = Nothing          ' Sicherstellen, dass es nicht noch einmal benutzt wird.
Else
    menuTable = CType(Session("MainMenuTable"), DataTable)
End If
```

Wir wollen hier keine ermüdende Diskussion über Sitzungsvariablen entfachen. In den Tagen von ASP sollte man Sitzungsvariablen besser meiden, weil sie die Skalierbarkeit einschränkten. Zum Glück ist diese Einschränkung bei ASP.NET verschwunden, außerdem stehen mehr Möglichkeiten für Sitzungsvariablen zur Verfügung. Sie müssen aber immer noch eine Balance zwischen Leistung und Skalierbarkeit herstellen. Egal, für welche Technik Sie sich entscheiden, die Balance ist ein Faktor. Wenn Sie nur an Leistung interessiert sind, wird die Skalierbarkeit leiden, und umgekehrt. Wir behandeln die Probleme mit Skalierbarkeit und Leistung genauer in Kapitel 11.

Zweitens müssen wir uns in unserem Benutzersteuerelement um die Zustandsverwaltung kümmern. Da wir von Seite zu Seite navigieren, müssen wir die Position des ausgewählten Menüregisters an die Seite übergeben. Auch in diesem Fall haben wir uns für eine Sitzungsvariable entschieden. Wir prüfen, ob die Sitzungsvariable `MainMenuIndex` bereits vorhanden ist. Ist das der Fall, tragen wir den Index aus der `DataList` in diese Sitzungsvariable ein. Andernfalls legen wir die Sitzungsvariable an und tragen den Wert 0 darin ein. So wird das erste Register aktiviert. (Wir nehmen an, dass Sie dies als Standardeinstellung wünschen. Falls nicht, können Sie den gewünschten Index ändern.)

```
If Session("MainMenuIndex") Is Nothing Then
    MainMenuList.SelectedIndex = 0
    Session("MainMenuIndex") = 0
Else
    MainMenuList.SelectedIndex = CInt(Session("MainMenuIndex"))
End If
```

Da wir eine Sitzungsvariable verwenden, lösen wir eine Verhaltensweise aus, der wir uns bewusst sein müssen. Falls die Sitzungsvariable ihre Verfallszeit erreicht, wird sie gelöscht. Der Client wird dann beim Hauptmenü in die Standardeinstellung zurückgeworfen. Verglichen mit einer Fehlermeldung über einen Timeout oder eine verlorene Verbindung ist das sogar ein Vorteil.

Wir dürfen auch nicht vergessen, den Zustand des Menüs beim Auswählen eines Registers zu sichern. Wird ein bestimmtes Register angeklickt, muss der entsprechende Wert in die weiter oben angelegte Sitzungsvariable eingetragen werden. Das passiert in `MenuList_ItemCommand`:

```
Private Sub MainMenuList_ItemCommand(ByVal source As Object,
    ByVal e As System.Web.UI.WebControls.DataListCommandEventArgs) _
    Handles MainMenuList.ItemCommand
    MainMenuList.SelectedIndex = e.Item.ItemIndex
    Session("MainMenuIndex") = e.Item.ItemIndex
    Response.Redirect(e.CommandArgument.ToString)
End Sub
```

Nachdem das Benutzersteuerelement `MainMenuTest` bereitsteht, können wir ein wirklich datengesteuertes Menü und die zugehörigen Seiten erstellen. Sie finden die Seite *Home.aspx* mit diesem Beispiel im Unterverzeichnis *MenuTabs*. Starten Sie die Datei *Home.aspx* und sehen Sie sich an, wie Sie das Menüregister von Seite zu Seite führt. Die Seiten *Home.aspx*, *Employees.aspx*, *Benefits.aspx* und *Company.aspx* enthalten in ihren Code-Behind-Dateien keinerlei Code für diese Aufgabe. Die Zustandsverwaltung erfolgt völlig unabhängig von den Webseiten. Sie brauchen lediglich das Benutzersteuerelement in die Webseiten einzufügen. Wenn Sie noch einmal zu Abbildung 7.4 zurückblättern, sehen Sie, dass wir bereits die Bereiche für das Logo, das Hauptmenü und den Seiteninhalt fertig haben. Jetzt können wir uns die vertikale Navigationsleiste vornehmen.

Das Benutzersteuerelement für die Navigationsleiste

Bevor wir uns mit der Navigationsleiste selbst beschäftigen, müssen wir einige Änderungen an dem Benutzersteuerelement für das Hauptmenü vornehmen. Die folgenden Beispiele finden Sie im Unterverzeichnis *MenuNavCombo*. Das Benutzersteuerelement für das Hauptmenü befindet sich in der Datei *MainMenu.ascx*. Dieses Steuerelement ist dem aus dem vorhergehenden Abschnitt sehr ähnlich. Wir haben allerdings den Code zum Erstellen der Menütabellen aus dem Benutzersteuerelement herausgenommen. So verbessern wir die Kapselung des Benutzersteuerelements und sorgen dafür, dass es universeller einsetzbar ist. Beachten Sie die Unterroutine `Page_Load` in der Code-Behind-Datei von *MainMenu.ascx*:

```
Imports MenuNavigation.LocalMenuData
⋮
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles MyBase.Load
    Dim menuData As New DataSet()
    Dim menuTable As New DataTable()
    If Session("MenuData") Is Nothing Then
        Dim getLocalMenu As New MenuDataClass()
        menuData = getLocalMenu.getMenuDataSet
        Session("MenuData") = menuData
        menuTable = menuData.Tables("MainMenuTable")
    Else
        menuTable = CType(Session("MenuData"), DataSet).Tables("MainMenuTable")
    End If
    ⋮
End Sub
```

Wir erstellen das `DataSet menuData` in dieser Code-Behind-Datei nicht mehr Zeile für Zeile, sondern rufen die `MenuDataClass` aus dem Namespace `MenuNavigation.LocalMenuData` auf. Diese Klasse füllt das `DataSet` und gibt es an das Benutzersteuerelement für das Hauptmenü zurück. Wir speichern das gesamte `DataSet` in dieser Sitzungsvariablen, nicht nur die Tabelle *MainMenuTable*. Das `DataSet` mit dem Hauptmenü enthält alle Tabellen, die für die Navigation benötigt werden. Da es sich um das Benutzersteuerelement für das Hauptmenü handelt, holen wir die Tabelle *MainMenuTable* aus dem `DataSet`:

```
menuTable = menuData.Tables("MainMenuTable")
```

Ist die Sitzungsvariable für das `DataSet menuData` bereits vorhanden, holen wir die Tabelle *MainMenuTable* direkt dort ab:

```
menuTable = CType(Session("MenuData"),DataSet).Tables("MainMenuTable")
```

Sehen wir uns kurz die Datei *MenuDataClass.vb* aus dem Verzeichnis *MenuNavigation* an. Die Datei definiert den Namespace `LocalMenuData`, und darin die Klasse `MenuDataClass`, die wir gerade in unserem Benutzersteuerelement aufgerufen haben. Diese Klasse kapselt die Menüerstellung. Sie ruft die Komponente `MenuDataServer` auf, übergibt Argumente zum Erstellen der Menü- und Navigationstabellen und gibt das fertige `DataSet` zurück. Indem wir diesen Code vom Benutzersteuerelement trennen, erhalten wir die gewünschte Kapselung. Wir vereinfachen auf diese Weise auch die Geschäftslogik zum Anpassen der Navigationsstruktur. Diese Geschäftslogik könnte die Berechtigungen des Benutzers auswerten, der sich bei der Anwendung angemeldet hat, und abhängig vom Ergebnis unterschiedliche Menü- und Navigationsstrukturen erstellen. Auch andere Anpassungen der Navigationsstruktur sind denkbar. Die Datei *MenuDataClass.vb* hat folgenden Inhalt:

MenuDataClass.vb

```
Imports MenuNavigation.MenuData
Imports System.Data
Namespace LocalMenuData
Public Class MenuDataClass
    Public Function getMenuDataSet() As DataSet
        ' Komponente MenuTable aufrufen und in die Tabelle MainMenuTable eintragen.
        Dim localMenuTables As New MenuDataServer()
        localMenuTables.AddMainMenuParameter(1, "Home", _
            "/MenuNavigation/MenuNavCombo/Home.aspx", False, _
            "Our Home Page")
        localMenuTables.AddMainMenuParameter(2, "Employees", _
            "/MenuNavigation/MenuNavCombo/Employees.aspx", True, _
            "All about Employees")
        localMenuTables.AddMainMenuParameter(3, "Benefits", _
            "/MenuNavigation/MenuNavCombo/Benefits.aspx", True, _
            "Our Company's Benefits")
        localMenuTables.AddMainMenuParameter(4, "Company", _
            "/MenuNavigation/MenuNavCombo/Company.aspx", False)
            'Add info to the NavBar Table
        localMenuTables.AddNavBarParameter(2, 1, "Address", _
            "/MenuNavigation/MenuNavCombo/Address.aspx", False)
        localMenuTables.AddNavBarParameter(2, 2, "Schedule", _
            "/MenuNavigation/MenuNavCombo/Schedule.aspx", False)
        localMenuTables.AddNavBarParameter(2, 3, "Emergency", -
            "/MenuNavigation/MenuNavCombo/Emergency.aspx", False)
        localMenuTables.AddNavBarParameter(3, 1, "Listing", _
            "/MenuNavigation/MenuNavCombo/Listing.aspx", False)
        Return localMenuTables.GetMenuDataSet
        localMenuTables.Dispose()      ' Objekt beseitigen.
        localMenuTables = Nothing      ' Sicherstellen, dass es nicht noch einmal verwendet wird.
    End Function
End Class
End Namespace
```

Der andere Unterschied bei der neuen Version des Steuerelements ist innerhalb von `MainMenuList_ItemCommand`. Statt einfach den Wert von `SelectedIndex` für die `DataList` einzustellen und seinen Zustand in der Sitzungsvariablen `MainMenuIndex` zu speichern, wollen wir auch den Zustand der Navigationsleiste übergeben. Dazu erstellen wir die Sitzungsvariable `NavBarIndex` und weisen ihr einen Wert zu. Wahrscheinlich fragen Sie sich, warum dieser Wert 100 ist. Wenn ein Benutzer ein Register des Hauptmenüs anklickt, sollte in der zugehörigen Navigationsleiste kein Eintrag als Standardeinstellung ausgewählt sein. Weil wir in den `ItemIndex` der Navigationsleiste den Wert 100 eintragen, kann das Benutzersteuerelement für die Navigationsleiste alle verfügbaren Möglichkeiten anbieten. (Wir setzen voraus, dass Sie nicht mehr als 99 Einträge haben. Sollte das doch der Fall sein, müssen Sie den Initialisierungswert eben auf 1000 erhöhen.)

```
Private Sub MainMenuList_ItemCommand(ByVal source As Object,
    ByVal e As System.Web.UI.WebControls.DataListCommandEventArgs)
    Handles MainMenuList.ItemCommand
    ainMenuList.SelectedIndex = e.Item.ItemIndex
    Session("MainMenuIndex") = e.Item.ItemIndex
    Session("NavBarIndex") = 100
    Response.Redirect(e.CommandArgument.ToString)
End Sub
```

Wir haben jetzt alles bereit, um das Benutzersteuerelement für die Navigationsleiste einsetzen zu können. Die entsprechende Beispieldatei ist *navbar.ascx*. Wenn Sie die Datei in Visual Studio .NET öffnen, werden Ihnen als Erstes die vertikale Ausrichtung und das `LinkButton`-Steuerelement `Menu` am oberen Rand auffallen. Wir haben auch eine HTML-Tabelle eingebaut, damit wir die `LinkButton`-Steuerelemente besser ausrichten und eine einheitliche Breite dafür einstellen können. Bevor wir den Code dieses Steuerelements genau erläutern, möchten wir erklären, warum wir ganz oben in der Navigationsleiste den `LinkButton` für das Menü eingebaut haben. Der erste Grund: Wenn wir die Bezeichnung des aktuellen Hauptmenüregisters als Überschrift in die seitliche Navigationsleiste eintragen, hilft das dem Benutzer festzustellen, wo er sich befindet. Das ist zwar schon ein hervorragender Grund, es gibt aber noch einen besseren: Falls Benutzer mit der Navigationsleiste arbeiten und zu der Hauptseite eines bestimmten Registereintrags zurückkehren wollen, müssten sie erneut auf das bereits ausgewählte Register im Hauptmenü klicken. Das ist nicht besonders intuitiv, daher verdrahten wir den ersten `LinkButton` der Navigationsleiste so mit dem Hauptmenü, dass er den Benutzer zu der Hauptseite des ausgewählten Hauptmenüregisters zurückbringt. Probieren Sie das selbst aus, indem Sie die Datei *home.aspx* aus dem Unterverzeichnis *MenuNavCombo* starten.

Sehen wir uns nun die Code-Behind-Datei von *Navbar.ascx* an. Der erste Teil des Ereignishandlers `Page_Load` lädt die korrekten Tabellen in das lokale `DataSet` und die Tabellen `menuTable` und `navTable`. Vielleicht fragen Sie sich, warum wir uns die Mühe machen, auch die Tabelle für das Hauptmenü zurückzugeben statt nur die für die Navigationsleiste. Wir tun das, weil wir feststellen müssen, welches Register im Hauptmenü ausgewählt ist. Nur so können wir ermitteln, ob es eine Navigationsleiste enthält. Außerdem brauchen wir seine Daten, um das vorhin erwähnte `DataLink`-Steuerelement `Menu` zu initialisieren.

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles MyBase.Load
    ' Hier Benutzercode zur Seiteninitialisierung einfügen.
    Dim menuData As New DataSet()
    Dim menuTable As New DataTable()   ' Eine DataTable-Instanz anlegen.
    Dim navTable As New DataTable()
    If Session("MenuData") Is Nothing Then
        Dim getLocalMenu As New MenuDataClass()
        menuData = getLocalMenu.getMenuDataSet
        Session("MenuData") = menuData
        menuTable = menuData.Tables("MainMenuTable")
        navTable = menuData.Tables("NavTable")
    Else
        navTable = CType(Session("MenuData"), DataSet).Tables("NavTable")
        menuTable = CType(Session("MenuData"), DataSet).Tables("MainMenuTable")
End If
```

Genau wie beim Benutzersteuerelement `MainMenu` überprüfen wir, ob die Sitzungsvariable `MenuData` vorhanden ist. Ist das nicht der Fall, legen wir sie an, andernfalls greifen wir einfach darauf zu. Wie schon vorher laden wir die Tabellen `menuTable` und `navTable`.

Danach ermitteln wir, ob das Benutzersteuerelement für die Navigationsleiste sichtbar ist. Diese Information steht in der `DataTable menuTable`. Wir holen uns die Information mit Hilfe der `Select`-Methode von ADO.NET aus der Tabelle. Während wir das Datenbankfeld *HasChild* abrufen, das festlegt, ob die Navigationsleiste sichtbar ist, holen wir auch gleich die Bezeichnung des ausgewählten Hauptmenüregisters und seinen URL. Diese Daten kommen in die Überschrift des Navigationsleisten-Steuerelements.

```
Dim privateNavBarSelected As Boolean
Dim selectString As String = "MenuID =" & CType(Session("MainMenuIndex"), Integer) + 1
Dim selectRow() As DataRow = menuTable.Select(selectString)
privateNavBarSelected = CType(selectRow(0)("HasChild"), Boolean)
Menu.Text = CType(selectRow(0)("MenuTabName"), String)
Menu.CommandArgument = CType(selectRow(0)("MenuTabURL"), String)
If privateNavBarSelected = False Then
    MyBase.Visible = False
    Exit Sub
Else
    MyBase.Visible = True
End If
```

Der `selectString` wird als ADO.NET-SQL-Anweisung erstellt. In diesem Fall trägt er in `MenuID` den Index des ausgewählten Hauptmenüregisters plus 1 ein. Wir müssen die 1 addieren, weil die `DataList` den Startindex 0 hat, die Hauptmenütabelle dagegen den Startindex 1. Die nächste Zeile erstellt eine neue Variable mit dem Namen `selectRow` und dem Typ `DataRow`. Dieser Variablen wird das Ergebnis der SQL-Anweisung `selectString` zugewiesen:

```
Dim selectRow() As DataRow = menuTable.Select(selectString)
```

Wir haben jetzt die Tabellenzeile, in der die Daten über das ausgewählte Register des Hauptmenüs gespeichert sind. Das Feld *HasChild* aus dieser Zeile enthält einen `Boolean`-Wert, der festlegt, ob die Navigationsleiste sichtbar ist. Wir werten diesen Wert später aus, um die Navigationsleiste sichtbar oder unsichtbar zu machen. Da wir ohnehin die komplette Zeile mit den Informationen über das Hauptmenüregister haben, weisen wir der Überschrift des Navigationsleisten-Benutzersteuerelements gleich den Namen und den URL des Hauptmenüregisters zu:

```
Menu.Text = CType(selectRow(0)("MenuTabName"), String)
Menu.CommandArgument = CType(selectRow(0)("MenuTabURL"), String)
```

Jetzt legen wir fest, ob das Benutzersteuerelement sichtbar sein soll. Dazu dient die Eigenschaft `MyBase.Visible`. Soll das Benutzersteuerelement unsichtbar sein, tragen wir in die Eigenschaft den Wert `False` ein und verlassen die Unterroutine sofort wieder. Es gibt keinen Grund, die restlichen Befehle auszuführen. Andernfalls tragen wir in `MyBase.Visible` den Wert `True` ein und fahren fort.

Wir müssen die verfügbaren Navigationsoptionen in der Navigationsleiste auf die Zeilen aus der Tabelle *navTable* beschränken, die mit dem zugehörigen Hauptmenüregister verknüpft sind. Dazu erstellen wir eine Sicht (view) auf die Tabelle *navTable* und filtern sie mit der *MenuID* des ausgewählten Hauptmenüregisters. Diese *MenuID* befindet sich noch in der Variablen `selectString`, die wir einige Zeilen weiter oben benutzt haben.

```
Dim navTableView As DataView = navTable.DefaultView
navTableView.RowFilter = selectString
NavBarList.DataSource = navTable.DefaultView
NavBarList.DataBind()
```

Dies lässt sich mit erstaunlich wenigen Codezeilen implementieren. Nachdem wir die `DefaultView`-Sicht für `navTable` in `navTableView` erstellt haben, weisen wir ihr einfach den `selectString` (`"MenuID=xxxx"`) als Zeilenfilter zu. Anschließend brauchen wir lediglich die Standardsicht an die `DataList` zu binden. Das war's, mehr brauchen wir nicht.

Der übrige Code gleicht dem aus dem Benutzersteuerelement `MainMenu`. Er erledigt die Zustandsverwaltung des Benutzersteuerelements. Probieren wir die beiden Benutzersteuerelemente für Hauptmenü und Navigationsleiste mal aus. Starten Sie *Home.aspx* aus dem Unterverzeichnis *MenuNavCombo*. Probieren Sie alle Kombinationen mit den Einträgen aus Hauptmenü und

Navigationsleisten aus. (Vergessen Sie nicht, dass Sie die Überschrift der Navigationsleiste anklicken können, um zur Hauptseite des aktuell ausgewählten Hauptmenüregisters zurückzukehren.)

Sehen wir uns ein paar Tricks an, mit denen wir die Benutzerfreundlichkeit erhöhen können. Wir haben sämtliche Seiten für dieses Beispiel erstellt, indem wir *Home.aspx* als Vorlage nahmen, sie kopierten und umbenannten. (Wir haben auch die Namenskonflikte in den Code-Behind-Dateien beseitigt, da beim Umbenennen lediglich die *.aspx*-Datei verändert wird. Die `@ Page`-Direktive mit der Anweisung `inherits="xxx"` bleibt gleich.) Nachdem wir die neue Seite fertig hatten, änderten wir ihren Inhalt und fügten ihren Namen in passende Hauptmenü- und Navigationsleisteneinträge in der Klasse `MenuDataClass` ein.

Das Aussehen der Benutzersteuerelemente wird teils durch Style Sheets und teils durch Deklarationen innerhalb des Benutzersteuerelements bestimmt. Beide haben wir uns bereits angesehen. Letztlich würden wir diese Einstellungen als Datenelemente verwirklichen, die von der Komponente `MenuData` zurückgegeben werden. Dann können wir sie genau wie die Tabellen für Hauptmenü und Navigationsleiste benutzen. Wir zeigen Ihnen ein Beispiel für diese Technik. Öffnen Sie die Code-Behind-Datei aus dem letzten Beispiel des Benutzersteuerelements *NavBar.ascx* und löschen Sie die Kommentarzeichen am Anfang der folgenden Zeilen:

```
' NavBarList.ItemStyle.BackColor = Color.Yellow
' NavBarList.ItemStyle.Font.Italic = True
' NavBarList.SelectedItemStyle.BackColor = Color.LightSeaGreen
' NavBarList.SelectedItemStyle.Font.Italic = True
' NavBarList.SelectedItemStyle.BorderWidth.Pixel(1)
' NavBarList.SelectedItemStyle.BorderStyle = BorderStyle.Ridge
```

Probieren Sie das Beispiel jetzt noch einmal aus. Sicher, es ist hässlich, aber es demonstriert unser neues Verfahren.

Weitere Optionen für Benutzersteuerelemente

Unsere Benutzersteuerelemente funktionieren zwar, es gibt aber einige Punkte, die sich verbessern lassen:

- **Die Benutzersteuerelemente MainMenu und NavBar in Serversteuerelemente verwandeln** Nach der Umwandlung in Serversteuerelemente könnten wir die Steuerelemente für Hauptmenü und Navigationsleiste in zahlreichen Anwendungen benutzen. Außerdem steht dann die Möglichkeit von Drag & Drop zur Verfügung. Wir empfehlen trotzdem, sie in anwendungsspezifische Benutzersteuerelemente zu ziehen. Auf diese Weise erhält das Benutzersteuerelement das Look-and-Feel der Anwendung und sie können es in eine Vorlage für die Anwendung einbinden. (In Kapitel 10 erfahren Sie mehr über Anwendungsvorlagen.)
- **Die Abstraktion verbessern, indem wir die Einstellungen für Farben, Schriftart und so weiter in die Komponente MenuData verlegen** Für diese Implementierung haben wir Ihnen bereits einen Ansatzpunkt gegeben. Falls sich herausstellt, dass diese Abstraktion wichtig ist, sollten Sie das gezeigte Verfahren nutzen und Ihre Steuerelemente entsprechend erweitern. Vergessen Sie nicht, den Einstellungen Standardwerte zuzuweisen.
- **Erweitern der Komponente MenuData, damit sie vorhandene Menünavigationstabellen oder XML verarbeitet** Das erreichen Sie durch zusätzliche öffentliche Konstruktoren in der Komponente `MenuData`, die die übergebenen Daten passend aufbereiten. Zum Beispiel könnte ein Konstruktor die benötigten Tabellen als Argumente erhalten. Die Komponente `MenuData` würde die Tabellen dann überprüfen und dem internen `DataSet` zuweisen, das die Menü- und Navigationsstrukturen speichert.

- **Weitere Navigationsebenen erstellen** Wenn Sie bei den Tabellen das Prinzip der 1:n-Beziehungen beibehalten und die Menüs miteinander verknüpfen, können Sie die Navigationsebenen so tief verschachteln, wie Sie es für sinnvoll halten. Vergessen Sie aber nicht, dass die Verwirrung des Benutzers mit jeder Navigationsebene zunimmt.
- **Grafiken für die Hauptmenüregister** Grafische Schaltflächen zu ermöglichen, ist nicht ganz so einfach. Sie brauchen dazu Funktionen, die eine Bitmap nehmen und sie passend zur Größe der Register vergrößern oder verkleinern. Das muss dynamisch geschehen, während die Komponenten die Navigationsstruktur erstellen. Eine andere Möglichkeit wäre, die maximal benötigte Länge der Register zu ermitteln, sie alle auf dieselbe Größe einzustellen und nur eine Grafik für ausgewählte beziehungsweise nicht ausgewählte Register zu übergeben. Wir haben solchen Code zwar entwickelt, es würde aber den Rahmen dieses Buchs sprengen, ihn in allen Einzelheiten vorzustellen.
- **Microsoft Internet Explorer-Websteuerelemente verwenden** Microsoft bietet Websteuerelemente an, die einen ähnlichen Funktionsumfang bieten wie unsere Benutzersteuerelemente. Sie sind nicht so abstrakt und lassen sich nicht so einfach an Daten binden, sie stellen aber eine Alternative dar. Wir haben festgestellt, dass wir eine feinere Steuerung benötigen, als diese Steuerelemente anbieten. Sie haben in diesem Kapitel die Arbeitsweise unserer Steuerelemente kennen gelernt und sind in der Lage, damit viel flexiblere Lösungen zu verwirklichen als mit Hilfe von Internet Explorer-Websteuerelementen. Aus diesem Grund haben wir kein Beispiel mit dieser Technik vorgestellt.

Ein Beispiel mit Visual Basic .NET-Windows Forms

Dieses Kapitel konzentriert sich zwar (wie das gesamte Buch) auf Visual Basic .NET-Komponenten für Webanwendungen, wir wollen aber trotzdem ein einfaches Beispiel für ein Windows Forms-Menü vorstellen. Diese Art Programm zu erstellen, ist im Vergleich mit älteren Visual Basic-Versionen viel einfacher geworden. Abbildung 7.7 zeigt das Ergebnis unserer Bemühungen.

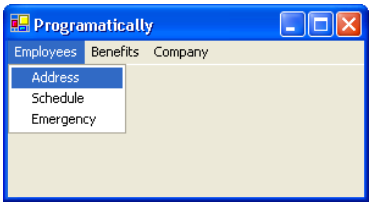

Abbildung 7.7: *Ein Beispiel für ein Windows Forms-Menü*

Sie finden die Projektmappe *WindowsFormsMenu* mit dem Beispiel im gleichnamigen Unterverzeichnis von *Ch07*. Öffnen Sie die Projektmappe und sehen Sie sich das Formular *ByHand.vb* an. Wir haben dieses Formular im Entwurfsmodus mit Hilfe des Menüeditors entworfen. Ziehen Sie über Drag & Drop eine `MainMenu`-Komponente aus der Toolbox, klicken Sie auf die Eigenschaft *Text* und fügen Sie Haupt- und Untereinträge hinzu. Visual Basic .NET generiert den entsprechenden Code im Codeausschnitt *Vom Windows Form Designer generierter Code*. Wenn Sie doppelt auf einen Menübefehl klicken, wird ein Ereignishandler für das `Click`-Ereignis dieses Menübefehls erzeugt. Dieser Ereignishandler sieht folgendermaßen aus:

```
Private Sub MenuItem2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles MenuItem2.Click
    Dim callAddress As New Address()    ' Dies ist das Click-Ereignis für den Befehl "Address".
    callAddress.Show()
End Sub
```

Wenn der Benutzer den Menübefehl *Address* auswählt, öffnet sich ein neues Formular.

Web Forms-Menüs lassen sich auch vom Programmcode aus erstellen. Ein Beispiel finden Sie in *ProgramTest.vb*. Ihre Code-Behind-Datei hat folgenden Inhalt:

```
Private localMainMenu As MainMenu
Private localMenuItem As MenuItem
Public Sub CreateMenu()
    localMainMenu = New MainMenu()
    localMenuItem = New MenuItem("&Employees")
    localMenuItem.MenuItems.Add("&Address", New System.EventHandler(AddressOf Me.MenuSelect))
    localMenuItem.MenuItems.Add("&Schedule", New System.EventHandler(AddressOf Me.MenuSelect))
    localMenuItem.MenuItems.Add("Eme&rgency")
    localMainMenu.MenuItems.Add(localMenuItem)
    ' Das neu erstellte Menü zum Menü dieses Formulars machen.
    Me.Menu = localMainMenu
End Sub
```

Zuerst legen wir Instanzen von `MainMenu` und `MenuItem` an. Wir erstellen ein neues `MenuItem`-Element und weisen ihm einen Namen zu:

```
localMenuItem = New MenuItem("&Employees")
```

Anschließend können wir Untereinträge in dieses `MenuItem` einfügen. Dazu rufen wir die Methode `MenuItems.Add` auf, der wir die Beschriftung und den Ereignishandler übergeben:

```
localMenuItem.MenuItems.Add("&Address", New System.EventHandler(AddressOf Me.MenuSelect))
```

Nachdem wir die Untereinträge zu dem `MainMenu` hinzugefügt haben, tragen wir dieses `MainMenu` als Menü des Formulars ein:

```
localMainMenu.MenuItems.Add(localMenuItem)
' Das neu erstellte Menü zum Menü dieses Formulars machen.
Me.Menu = localMainMenu
```

Das Aussehen des Menüs ist damit festgelegt. Jetzt müssen wir die Ereignishandler zuordnen. Eigentlich sollte dieser Schritt vor der Unterroutine `CreateMenu` erfolgen. Andernfalls werden Sie im Code-Editor bei `System.EventHandler(AddressOfMe.MenuSelect)` auf Fehler hingewiesen. Denken Sie daran, falls Sie selbst eine solche Menüstruktur entwerfen. Der folgende Ereignishandler öffnet ein Meldungsfeld, wenn ein Menübefehl angeklickt wird:

```
Protected Sub MenuSelect(ByVal sender As Object, ByVal e As System.EventArgs)
    MessageBox.Show("You Clicked a Menu Button")
End Sub
```

Wenn Sie in unserem Beispiel auf eine Schaltfläche drücken, wird die gezeigte Unterroutine `CreateMenu` vom `Click`-Ereignishandler der Schaltfläche aufgerufen und erzeugt dynamisch das Menü. Probieren Sie das Programm aus und wählen Sie die verschiedenen Menübefehle.

Wir wollen dieses Beispiel abstrahieren und auf ähnliche Weise kapseln wie die Web Forms-Klassen aus dem ersten Teil dieses Kapitels. Den entsprechenden Code finden Sie in der Datei *Programmatically.vb*. Wenn Sie sich die Code-Behind-Datei dieser Seite ansehen, werden Sie feststellen, dass sie nur sehr wenig zusätzlichen Code enthält. Zuerst erstellen wir den Ereignishandler für das Menü, obwohl das Menü in einer anderen Klassendatei angelegt wird.

```
Protected Sub MenuSelect(ByVal sender As Object, ByVal e As System.EventArgs)
    MessageBox.Show("You Clicked a Menu Button")
End Sub
```

Die Klasse `MenuTest`, die das Menü erstellt, wird im Codeabschnitt *Vom Windows Form Designer generierter Code* aufgerufen. Sie finden den entsprechenden Code in der Methode `New`:

```
Public Sub New()
    MyBase.New()
    ' Dieser Aufruf ist für den Windows Form-Designer erforderlich.
    InitializeComponent()
    ' Initialisierungen nach dem Aufruf InitializeComponent() hinzufügen.
    Dim localMenu As New MenuTest()
    Me.Menu = localMenu.getMenuStructure
End Sub
```

Hier wird die Klasse `MenuTest` aufgerufen. Bevor wir sie benutzen können, müssen wir die Anweisung `Imports WindowsFormsMenu.GetMenuTest` einfügen. Die Klasse `MenuTest` hat eine einzige Methode namens `getMenuStructure`. Sie gibt das Menü in Form des Typs `MainMenu` zurück, das der Seiteneigenschaft `Me.Menu` zugewiesen wird. In diesem Beispiel wird das Menü erstellt, während die Instanz des Formulars angelegt wird. Auf diese Weise ist es einfacher, das Formular als Basisklasse für andere Formulare der Anwendung zu benutzen, die alle automatisch dieselbe Menüstruktur erhalten (genauer gesagt: erben).

Sehen wir uns die Klasse `MenuTest` genauer an:

MenuTest.vb

```
Namespace GetMenuTest
Public Class MenuTest
    Private privateMainMenu As MainMenu
    Private privateMenuItem As MenuItem
    Protected Sub MenuSelect(ByVal sender As Object, ByVal e As System.EventArgs)
        MessageBox.Show("You Clicked a Menu Button")
    End Sub
    Public Function getMenuStructure() As MainMenu
        privateMainMenu = New MainMenu()
        privateMenuItem = New MenuItem("&Employees")
        privateMenuItem.MenuItems.Add("&Address", New System.EventHandler(AddressOf Me.MenuSelect))
        privateMenuItem.MenuItems.Add("&Schedule")
        privateMenuItem.MenuItems.Add("Eme&rgency")
        privateMainMenu.MenuItems.Add(privateMenuItem)
        privateMenuItem = New MenuItem("&Benefits")
        privateMenuItem.MenuItems.Add("&Listing")
        privateMainMenu.MenuItems.Add(privateMenuItem)
        privateMenuItem = New MenuItem("&Company")
        privateMainMenu.MenuItems.Add(privateMenuItem)
        Return privateMainMenu
    End Function
End Class
End Namespace
```

Der Code ist geradlinig, er nutzt die Funktionen, die wir vorher in der Code-Behind-Datei des Formulars implementiert haben.

Wir verfügen nun über eine funktionsfähige, datengesteuerte Menüstruktur für Windows Forms. Wir könnten sie noch dadurch erweitern, dass wir die Menüeinträge auf der Basis von Datenbankelementen erstellen, wie in unserem Web Forms-Beispiel.

Zusammenfassung

Eine durchdachte Menü- und Navigationsstruktur ist eine wichtige Eigenschaft jeder Windows- oder Webanwendung. Eine strukturierte Navigationsmethode, mit der sich der Benutzer durch die Seiten bewegen kann, erhöht die Benutzerfreundlichkeit unserer Anwendungen. Sogar noch wichtiger ist, datengesteuerte Komponenten zu entwickeln, die automatisch die Menü- und Navigationsstruktur unserer Anwendung generieren. Auf diese Weise können wir unsere Anwendungen schnell zusammenstellen, und sie werden ein einheitliches Aussehen haben. Aufbau und Reihenfolge der Menüs lassen sich schnell verändern, wir können einfach neue Seiten hinzufügen. Noch wichtiger ist die Fähigkeit, abhängig von der Rolle des Benutzers unterschiedliche Menü- und Navigationsstrukturen zu erzeugen und festzulegen zu können, welche Menü- und Navigationselemente sichtbar oder funktionsfähig sind. Das sollte außerdem ohne komplizierte Programmierung möglich sein. Dank unserer Anstrengungen haben wir wieder verwendbare Standardkomponenten, die sich um Hauptmenüregister und Navigationsleiste kümmern. Wir haben ihre Funktionsweise so genau erklärt, dass Sie die Komponenten sofort in Ihren eigenen Anwendungen einsetzen oder sie an Ihre jeweiligen Anforderungen anpassen können.

In den bisherigen Kapiteln haben wir universelle, wieder verwendbare Objekte und Komponenten entwickelt. Im nächsten Kapitel beschäftigen wir uns mit der Geschäftsschicht. Hier laufen die Fäden für alle anwendungsspezifischen Funktionen und Prozesse zusammen.