Kapitel 6

Grundlagen von Webparts

In diesem Kapitel:

Wenn Sie einen Microsoft SharePoint 2010-Entwickler bitten, eines der wichtigsten Features zu nennen, antwortet er wahrscheinlich: »Natürlich Webparts«. Aber was sind Webparts? Es sind im Grunde nur anpassbare Bereiche, die innerhalb einer SharePoint-Webseite gehostet werden. Webparts wurden bereits vor vielen Jahren in Microsoft SharePoint Team Services 2001 eingeführt. In den nachfolgenden Versionen des Produkts wurde das Konzept der Webparts verfeinert und hat sich weit verbreitet. In Microsoft .NET 2.0 verschob sich die Infrastruktur für Webparts von SharePoint in die offizielle ASP.NET-Webentwicklungsplattform, sodass die Verwendung von Webparts in vielen unterschiedlichen ASP.NET-Anwendungen umfassend unterstützt wurde. Aus Sicht des Endbenutzers ist ein Webpart lediglich ein Abschnitt einer Webseite, den der Benutzer selbst über die Webbrowseroberfläche anpassen kann. Aus Sicht des Entwicklers ist ein Webpart eine Klasse, die Code zum Darstellen ihres Inhalts im Browser und zum Verarbeiten einer benutzerdefinierten Konfiguration, des Layouts, der Anordnung und so weiter, innerhalb der SharePoint- und/oder ASP.NET-Umgebung definiert. Der Benutzer kann Webparts selbstständig in Seiten (den sogenannten Webpartseiten) hinzufügen und entfernen, indem er sie aus einer Servergalerie oder einer öffentlichen Onlinegalerie auswählt.

Wichtiger aus Sicht des Entwicklers ist, dass Webparts in vielen unterschiedlichen Seiten und Websites wiederverwendet werden können. Das vereinfacht die Entwicklung benutzerdefinierter Lösungen, die Bereitstellung und die Wartung. Viele SharePoint-Lösungen basieren auf benutzerdefinierten Webparts, die in Webpartseiten benutzt werden.

Dieses Kapitel erklärt, wie Webparts funktionieren und wie Sie eigene Webparts entwickeln. Kapitel 7, »Fortgeschrittene Webparts«, setzt das Thema fort und beschreibt fortgeschrittene Techniken bei der Webpartentwicklung.

Architektur von Webparts

Ein Webpart ist ein benutzerdefiniertes ASP.NET-Steuerelement, das von der Basisklasse *WebPart* aus dem Namespace *System.Web.UI.WebControls.WebParts* abgeleitet ist. Damit ein Webpart in einer Seite uneingeschränkt genutzt werden kann, müssen Sie ein *WebPartZone*-Steuerelement definieren, einen Container für mehrere Webparts. Das Steuerelement *WebPartZone* stellt ein gemeinsames Darstellungsmuster für alle enthaltenen Webparts zur Verfügung. Ein anderes zentrales Steuerelement in der Architektur von Webparts ist *WebPartManager*, das alle Aufgaben im Zusammenhang mit der Lebensdauerverwaltung von Webparts übernimmt, beispielsweise das Laden/Entladen und Serialisieren/Deserialisieren ihres Zustands innerhalb der aktuellen Seite und das Zusammenfassen von Webparts zu Webpartzonen. SharePoint hat eigene *WebPartZone*-Steuerelemente, mit denen Sie die Möglichkeit erhalten, SharePoint-spezifische Darstellungszonen zu definieren. Einige Beispiele sind die Klasse *WebPartZone* für die Standarddarstellung von Webparts und die Klasse *EditorZone* für die Darstellung von Webparts zum Editieren anderer Webparts (mehr zu diesen Editorwebparts weiter unten in diesem Kapitel). Außerdem wurde für das Steuerelement *WebPartManager* in SharePoint eine angepasste Implementierung namens *SPWebPartManager* erstellt; sie verarbeitet bestimmte Aktionen, die nur in SharePoint zur Verfügung stehen. Um diese Steuerelemente nutzen zu können, stellt SharePoint auch noch den benutzerdefinierten Seitentyp *WebPartPage* aus dem Namespace *Microsoft.SharePoint.WebPartPages* zur Verfügung. Er enthält eine vorkonfigurierte und eindeutige Instanz eines *SPWebPartManager*-Steuerelements und die Hauptwebpartzonen, die nützlich sind, um eine Seite darzustellen, die sich auf Webparts aufbaut. Abbildung 6.1 zeigt, wie eine solche Seite aufgebaut ist.

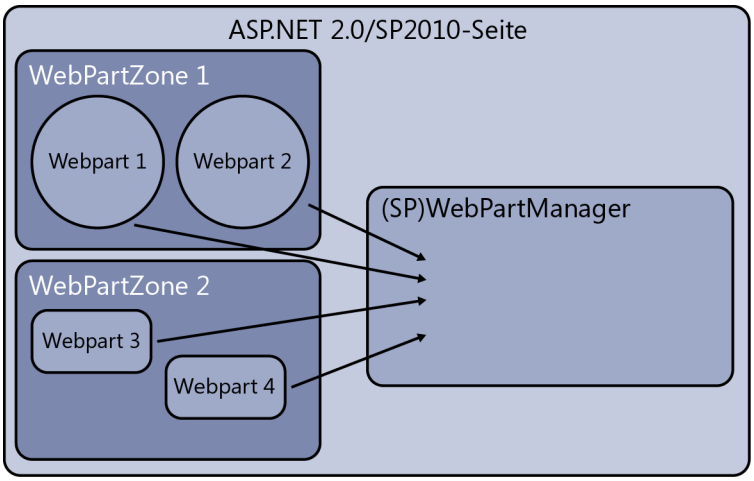

Abbildung 6.1 Architektur einer *WebPartPage* in SharePoint und ASP.NET

In Ihren Lösungen arbeiten Sie in erster Linie mit Webparts. Mit Webpartzonen und *WebPartManager* werden Sie nur selten direkt zu tun haben.

Ein »Hallo, Welt«-Webpart

Jetzt ist es an der Zeit, dass Sie Ihr erstes Webpart entwickeln. Microsoft Visual Studio 2010 stellt einige Projektvorlagen und Dienstprogramme bereit, die Ihnen helfen, benutzerdefinierte Webparts schnell zu entwickeln. Nehmen wir an, Sie brauchen ein »Hallo, Welt«-Webpart, das den aktuellen Benutzer einfach begrüßt, und seinen Namen und die aktuelle Zeit im Browser ausgibt. Sie beginnen damit, dass Sie ein neues Projekt vom Typ *SharePoint/2010/Leeres SharePoint-Projekt* anlegen. Diese Projektvorlage enthält nur einige Assemblyverweise. Sie ist nützlich, um beliebige SharePoint-Lösungen mit einer vordefinierten Bereitstellungskonfiguration zu entwickeln. Wenn Sie ein neues SharePoint-Projekt anlegen, fordert Visual Studio die URL der Website an, in der es die Lösung bereitstellt. Außerdem geben Sie an, welche Art von Bereitstellung Sie erstellen (Farmlösung oder Sandkastenlösung). Wählen Sie in diesem Beispiel die Option *Als Farmlösung bereitstellen*. Über die Bereitstellung erfahren Sie weiter unten in diesem Kapitel im Abschnitt »Bereitstellen von Webparts« mehr, und Kapitel 8, »SharePoint-Features und -Lösungen«, befasst sich ausführlich mit dem Thema. Vorerst sollten Sie sich auf das Webpart selbst konzentrieren.

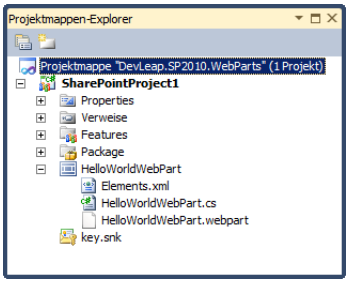

Abbildung 6.2 Inhalt des Projekts mit dem Beispielwebpart

Um Ihr Beispielwebpart zu entwickeln, müssen Sie ein neues Dateielement vom Typ *Webpart* zum Projekt hinzufügen. Nennen Sie das neue Element *HelloWorldWebPart*. Daraufhin wird eine neue Klassendatei hinzugefügt, zusammen mit einem Satz von Konfigurationsdateien, die ich später beschreibe. Abbildung 6.2 zeigt den Inhalt des Projekts, nachdem Sie das Webpartelement hinzugefügt haben.

Listing 6.1 zeigt den Inhalt der Datei *HelloWorldWebPart.cs*, unmittelbar nachdem Sie das Webpartelement zum Projekt hinzugefügt haben.

Listing 6.1 Die anfängliche Klassendatei für das »Hallo, Welt«-Webpart

```
using System;
using System.ComponentModel;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using Microsoft.SharePoint;
using Microsoft.SharePoint.WebControls;

namespace DevLeap.SP2010.WebParts.HelloWorldWebPart {
    [ToolboxItemAttribute(false)]
    public class HelloWorldWebPart : WebPart  {
        protected override void CreateChildControls()
        {
        }
    }
}
```

In diesem Code fällt als Erstes auf, dass die Klasse von der Basisklasse *WebPart* abgeleitet ist, wie bereits im vorherigen Abschnitt erwähnt. Die wichtigste Stelle im Beispielcode ist aber die Überschreibung der Methode *CreateChildControls*. Wie bei jedem anderen benutzerdefinierten ASP.NET-Steuerelement sollten Sie hier die Struktur des Websteuerelements erzeugen, die festlegt, wie das Webpart dargestellt wird. Listing 6.2 fügt einige Instanzen von *LiteralControl* hinzu, um die Begrüßungsmeldung in einem *H1*-Tag und die aktuelle Zeit in einem *DIV*-Element anzuzeigen.

Listing 6.2 Der Code für das »Hallo, Welt«-Webpart

```
using System;
using System.ComponentModel;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using Microsoft.SharePoint;
using Microsoft.SharePoint.WebControls;

namespace DevLeap.SP2010.WebParts.HelloWorldWebPart {
    [ToolboxItemAttribute(false)]
    public class HelloWorldWebPart : WebPart  {
        protected override void CreateChildControls()
```

```
        {
            SPWeb currentWeb = SPControl.GetContextWeb(HttpContext.Current);
            String currentUserName = currentWeb.CurrentUser.LoginName;

            // "Willkommen"
            this.Controls.Add(new LiteralControl(String.Format(
                "<h1>Welcome {0}!</h1>", currentUserName)));
            // "Aktuelle Zeit"
            this.Controls.Add(new LiteralControl(String.Format(
                "<div>Current DateTime: {0}</div>", DateTime.Now)));
        }
    }
}
```

Am Anfang der Methode *CreateChildControls* fordert der Code die aktuelle *SPWeb*-Instanz von der Klasse *SPControl* an. Dabei verwendet er die aktuelle *HttpContext*-Instanz, damit er den Anmeldenamen (*LoginName*) des aktuellen Benutzers abrufen kann.

WEITERE INFORMATIONEN Einzelheiten über die Klassen *SPWeb* und *SPControl* finden Sie in Kapitel 3, »Serverobjektmodell«.

Wie Sie in diesem einführenden Beispiel sehen, muss ein guter Webpartentwickler zuerst einmal ein guter ASP.NET-Entwickler sein. Und ein ASP.NET-Entwickler wird keine Probleme haben, Webparts zu entwickeln.

Abbildung 6.3 zeigt die Ausgabe des »Hallo, Welt«-Webparts, wenn es in die Homepage einer Webanwendung mit forderungsbasierter Authentifizierung eingefügt wurde.

WEITERE INFORMATIONEN Einzelheiten über die forderungsbasierte Authentifizierung finden Sie in Kapitel 22, »Forderungsbasierte Authentifizierung und Identitätsverbunde«.

HINWEIS Eine andere Möglichkeit, ein Webpart zu implementieren, besteht darin, es von der Klasse *WebPart* aus dem Namespace *Microsoft.SharePoint.WebPartPages* abzuleiten. Diese Klasse ist intern allerdings von der ASP.NET-Basisklasse *WebPart* abgeleitet, die in erster Linie die Abwärtskompatibilität zu älteren Versionen von Microsoft SharePoint sicherstellen soll. Wenn Sie Ihre Webparts von der SharePoint-Basisklasse *WebPart* ableiten, funktionieren sie nur in SharePoint-Websites, nicht in Standard-ASP.NET-Websites. Verwenden Sie dagegen die benutzerdefinierte SharePoint-Basisklasse, können Sie einige zusätzliche Funktionen nutzen, die in der Infrastruktur für Standardwebparts nicht zur Verfügung stehen. Wir haben allerdings die Erfahrung gemacht, dass diese zusätzlichen Fähigkeiten nicht wirklich nützlich sind. Dennoch zählen wir am Ende dieses Kapitels die wenigen Vorteile dieser Art von Webparts auf.

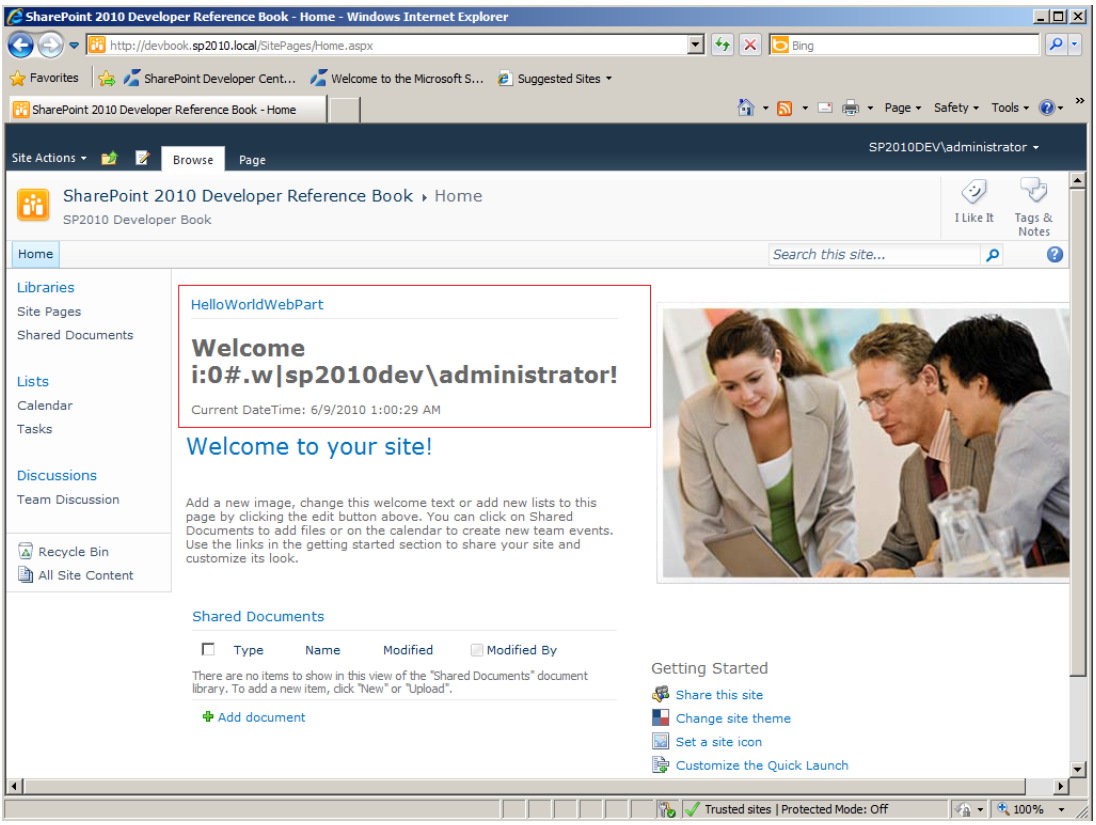

Abbildung 6.3 Die Ausgabe von *HelloWorldWebPart* innerhalb einer SharePoint 2010-Website mit forderungsbasierter Authentifizierung

Bereitstellen von Webparts

Um das Beispielwebpart bereitzustellen, gehen Sie (wie bei jeder Webpartimplementierung) folgendermaßen vor:

- Erstellen Sie aus der Klasse eine .NET-Assembly vom Typ DLL.
- Machen Sie die Assembly für die Webanwendung verfügbar, indem Sie sie in den GAC, den lokalen *bin*-Ordner der Webanwendung oder den Lösungskatalog der aktuellen Websitesammlung legen.

> **HINWEIS** GAC steht für Globally Assembly Cache, ein zentrales, gemeinsam genutztes Repository vertrauenswürdiger und digital signierter .NET-Assemblys. Einzelheiten über die .NET-Entwicklung und Bereitstellung finden Sie im Buch *Applied Microsoft .NET Framework Programming* von Jeffrey Richter (Microsoft Press 2002, ISBN 978-0-7356-1422-2).

- Autorisieren Sie das Webpart für die Ausführung innerhalb der aktuellen SharePoint-Umgebung.
- Laden Sie das Webpart in den Webpartkatalog der aktuellen Website, damit es den Endbenutzern zur Verfügung steht.

Visual Studio 2010 macht es einfach, all diese Bereitstellungsschritte auszuführen. Wählen Sie einfach den Menübefehl *Erstellen/<Lösung> bereitstellen*, dann wird das Webpart automatisch auf der Website bereitgestellt, die Sie beim Anlegen des Projekts konfiguriert haben.

Sehen wir uns an, wie diese Schritte im Detail aussehen. Das Erstellen der .NET-Assembly ist trivial, daher gehe ich nicht weiter darauf ein. Sie müssen aber darauf achten, dass Sie jedem Objekt, das Sie in den GAC legen wollen, einen starken Namen (Name, Version, Kultur und Token des öffentlichen Schlüssels) zuweisen müssen. Glücklicherweise erledigt Visual Studio 2010 das für Sie, indem es automatisch Signaturschlüssel zum Projekt hinzufügt. Auch die Assembly in den GAC oder den *bin*-Ordner der Webanwendung zu legen ist für einen .NET-Entwickler simpel. Wollen Sie dagegen die Assembly im Lösungskatalog der aktuellen Websitesammlung installieren, müssen Sie sich mit Sandkastenlösungen auskennen, daher gehe ich darauf später in Kapitel 8 genauer ein.

Um das Webpart so zu autorisieren, dass es innerhalb der SharePoint-Umgebung ausgeführt werden darf, müssen Sie ein spezielles Konfigurationselement in die *Web.config*-Datei der aktuellen Webanwendung einfügen und damit das Webpart zu einem »sicheren Steuerelement« (*SafeControl*) erklären. Am Ende von Kapitel 7 erfahren Sie mehr über *SafeControl*. Listing 6.3 zeigt einen Ausschnitt der benutzerdefinierten Konfiguration, die Sie anwenden müssen.

Listing 6.3 Die benutzerdefinierte Konfiguration bewirkt, dass das »Hallo, Welt«-Webpart als sicher für SharePoint erklärt wird

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<configuration>
  <SharePoint>
    <!-- Aus Platzgründen entfernt -->
    <SafeControls>
      <!-- Hier stehen viele andere SafeControl-Elemente -->
      <SafeControl Assembly="DevLeap.SP2010.WebParts, Version=1.0.0.0,
      Culture=neutral, PublicKeyToken=cba640f292988abf"
      Namespace="DevLeap.SP2010.WebParts.HelloWorldWebPart" TypeName="*" Safe="True"
      SafeAgainstScript="False" />
    </SafeControls>
    <!-- Aus Platzgründen entfernt -->
</configuration>
```

Sie müssen die Webpartdefinition zur aktuellen Websitesammlung hinzufügen, um das Webpart im Webpartkatalog verfügbar zu machen. Diese Definition ist eine *.webpart*-Datei, die Visual Studio 2010 automatisch generiert, sobald Sie ein Webpartelement zum Projekt hinzufügen. Listing 6.4 zeigt den Standardinhalt dieser Datei in unserem Beispiel.

Listing 6.4 Die *.webpart*-Datei zum Bereitstellen des »Hallo, Welt«-Webparts

```
<?xml version="1.0" encoding="utf-8"?>
<webParts>
  <webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
    <metaData>
      <type name="DevLeap.SP2010.WebParts.HelloWorldWebPart.HelloWorldWebPart,
        $SharePoint.Project.AssemblyFullName$" />
      <importErrorMessage>$Resources:core,ImportErrorMessage;</importErrorMessage>
    </metaData>
```

```
    <data>
      <properties>
        <property name="Title" type="string">HelloWorldWebPart</property>
        <property name="Description" type="string">My WebPart</property>
      </properties>
    </data>
  </webPart>
</webParts>
```

Die wesentliche Stelle in der *.webpart*-Datei ist die Deklaration des Typs (eines .NET-Typs), der dem aktuellen Webpart zugeordnet ist. Beachten Sie, dass eine *.webpart*-Datei viele Webparts deklarieren kann, auch wenn Visual Studio 2010 standardmäßig für jede Webpartdefinition eine eigene *.webpart*-Datei anlegt. Der Typname unseres »Hallo, Welt«-Webparts ist als vollständiger Name (Namespace + Klassenname) deklariert, ergänzt durch den Namen der Assembly, in dem es enthalten ist. In diesem Codebeispiel ist der Assemblyname mit einem Alias definiert *($SharePoint.Project.AssemblyFullName$)*, den Visual Studio 2010 während des Bereitstellungsprozesses automatisch durch den tatsächlichen Assemblynamen ersetzt.

Außerdem deklariert die *.webpart*-Datei die Standardwerte für einige Eigenschaften des Webparts. Sie sehen beispielsweise, dass die Eigenschaften *Title* und *Description* des Webparts als benutzerdefinierte *property*-Elemente innerhalb eines übergeordneten *properties*-Elements definiert sind.

Sie können die Werte dieser Eigenschaften ändern und einige andere Eigenschaften definieren, indem Sie die *.webpart*-Datei in Visual Studio editieren. Tabelle 6.1 listet die nützlichsten Eigenschaften auf, die Sie definieren können.

Tabelle 6.1 Wichtige konfigurierbare Eigenschaften in einer *.webpart*-Datei

Eigenschaftsname	Beschreibung
Title	Legt den Titel des Webparts fest. Der Endbenutzer bekommt den Titel im Webpartkatalog angezeigt und wenn er ein Webpart in eine Seite einfügt. Außerdem wird dies der Standardtitel eines neu eingefügten Webparts.
Description	Eine Beschreibung des aktuellen Webparts. Sie wird dem Endbenutzer im Webpartkatalog angezeigt und wenn er ein Webpart in eine Seite einfügt.
TitleIconImageUrl	Gibt die URL für ein Bild an, mit dem das Webpart in seiner Titelzeile dargestellt wird. Der Standardwert ist eine leere Zeichenfolge ("").
CatalogIconImageUrl	Gibt die URL für ein Bild an, mit dem das Webpart im Webpartkatalog dargestellt wird. Der Standardwert ist eine leere Zeichenfolge ("").
ChromeType	Legt den Typ des Rahmens um das Webpart fest. Diese Eigenschaft kann folgende Werte annehmen (der Standardwert ist *Default)*: ■ *Default* Das Verhalten der übergeordneten Webpartzone wird übernommen. ■ *TitleAndBorder* Eine Titelzeile mit einem Rand. ■ *None* Kein Rand und keine Titelzeile. ■ *TitleOnly* Eine Titelzeile ohne Rand. ■ *BorderOnly* Ein Rand ohne Titelzeile.
ChromeState	Legt fest, ob das Webpart im Modus *Minimized* oder *Normal* angezeigt wird.
AllowClose	Steuert, ob das Webpart von einem Endbenutzer geschlossen werden kann.
AllowConnect	Steuert, ob das Webpart von einem Endbenutzer mit einem anderen verbunden werden darf. ▶

Eigenschaftsname	Beschreibung
AllowEdit	Steuert, ob das Webpart von einem Endbenutzer bearbeitet werden kann.
AllowHide	Steuert, ob das Webpart von einem Endbenutzer verborgen werden darf.
AllowMinimize	Steuert, ob das Webpart von einem Endbenutzer minimiert werden kann.
AllowZoneChange	Steuert, ob das Webpart von einem Endbenutzer zwischen unterschiedlichen Webpartzonen verschoben werden darf.
ExportMode	Legt fest, ob die Konfiguration des aktuellen Webparts zur Wiederverwendung in einer anderen Website exportiert werden kann.

Listing 6.5 demonstriert, wie ich die *.webpart*-Datei für das »Hallo, Welt«-Beispielwebpart angepasst habe.

Listing 6.5 Die *.webpart*-Datei zum Bereitstellen des konfigurierten »Hallo, Welt«-Webparts

```
<?xml version="1.0" encoding="utf-8"?>
<webParts>
  <webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
    <metaData>
      <type name="DevLeap.SP2010.WebParts.HelloWorldWebPart.HelloWorldWebPart,
          $SharePoint.Project.AssemblyFullName$" />
      <importErrorMessage>$Resources:core,ImportErrorMessage;</importErrorMessage>
    </metaData>
    <data>
      <properties>
        <property name="Title" type="string">HelloWorldWebPart</property>
        <property name="Description" type="string">
          Custom WebPart to welcome end user</property>
        <property name="CatalogIconImageUrl"
          type="string">/_layouts/images/ICTXT.GIF</property>
        <property name="AllowEdit" type="bool">true</property>
        <property name="ChromeType" type="chrometype">TitleAndBorder</property>
      </properties>
    </data>
  </webPart>
</webParts>
```

Abbildung 6.4 zeigt, wie dieses angepasste »Hallo, Welt«-Webpart dargestellt wird. Beachten Sie die benutzerdefinierte Kategorie, das Symbol im Webpartkatalog, die angepasste Beschreibung und die Randdarstellung (*TitleAndBorder*).

Wie Sie im Abschnitt »Konfigurierbare Webparts« weiter unten in diesem Kapitel sehen, können Webparts auch benutzerdefinierte Eigenschaften haben, die der Entwickler definiert und die von Websitebesitzern oder -mitgliedern verändert werden dürfen, sofern sie über ausreichende Berechtigungen verfügen. Solche Eigenschaften können Sie bei der Bereitstellung der Webparts mit Standardwerten konfigurieren, genau wie die gerade vorgestellten Standardwebparteigenschaften.

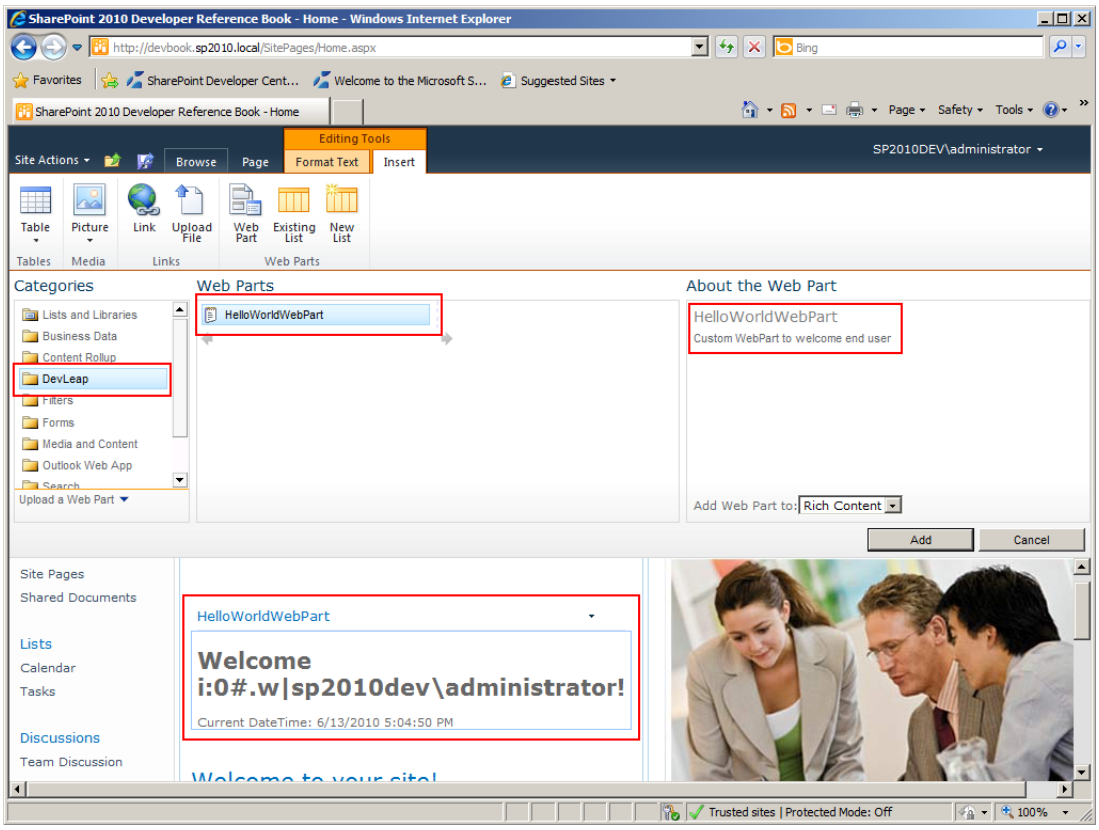

Abbildung 6.4 Das angepasste Steuerelement *HelloWorldWebPart* in einer SharePoint 2010-Website

Webparts in Unternehmenslösungen

Sie haben im letzten Abschnitt am Beispiel des »Hallo, Welt«-Webparts gesehen, wie Sie ein ganz simples Webpart definieren und bereitstellen. Natürlich sind Webparts in der Praxis etwas komplexer, sie umfassen mehr Steuerelemente und ihr Verhalten ist aufwendiger zu definieren. In diesem Abschnitt lernen Sie zwei Arten von Webparts kennen: klassische Webparts, die aus eigenem Code erstellt werden, und visuelle Webparts, die im grafischen Designer von Visual Studio 2010 entworfen werden.

Klassische Webparts

Ein klassisches Webpart (classic web part) ist ein Steuerelement, das aus mehreren ASP.NET-Steuerelementen besteht. Es tritt mit dem Endbenutzer über Ereignisse und Steuerelementverhalten in Interaktion. In diesem Abschnitt erstellen Sie ein Webpart für die Dateneingabe, das Daten vom Endbenutzer entgegennimmt und in eine SharePoint-Zielliste einfügt. Der Kern dieses Webparts nutzt das SharePoint-Serverobjektmodell, um die Elemente in die Zielliste einzufügen. Die Benutzeroberfläche wird aus ASP.NET-Serversteuerelementen aufgebaut.

Nehmen wir an, Sie haben in Ihrer SharePoint-Website eine Zielliste mit Kontaktanfragen (*Requests for Contacts*) und wollen die Anfragen der Benutzer mit Ihrem benutzerdefinierten Webpart eingeben lassen. Abbildung 6.5 zeigt das fertige Webpart.

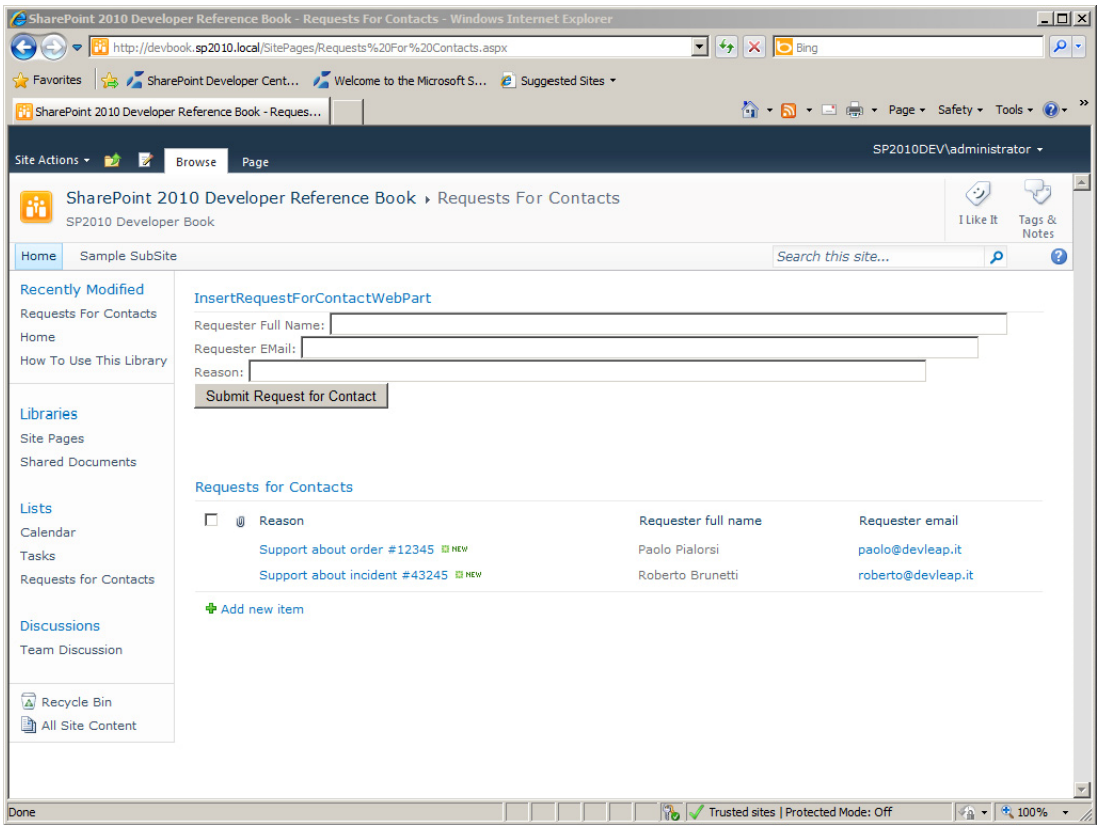

Abbildung 6.5 Das Webpart *InsertRequestForContactWebPart* in einer SharePoint 2010-Website

Nennen Sie das Webpart *InsertRequestForContactWebPart* und legen Sie in Visual Studio 2010 ein SharePoint-Projekt dafür an. Wählen Sie die Option *Farmlösung* als Projekttyp. Das Webpart stellt einige Felder zur Verfügung (Grund für die Kontaktanforderung, vollständiger Name und E-Mail des Benutzers), in denen die Anfrage beschrieben wird. Diese Felder werden auf die Zielliste *Requests for Contacts* abgebildet, die Sie von Hand in der aktuellen Website definiert haben.

HINWEIS In Kapitel 10, »Bereitstellen von Daten«, erfahren Sie, wie Sie vom Programmcode aus Datenstrukturen wie die Liste *Requests for Contacts* definieren und bereitstellen. In einer Unternehmenslösung müssen Sie wahrscheinlich die Liste und die darin eingesetzten Webparts innerhalb einer gemeinsamen SharePoint-Lösung definieren, die sie dann »auf einen Schlag« bereitstellen.

Intern umfasst das Webpart mehrere ASP.NET-Steuerelemente für die Eingabefelder. Es arbeitet mit dem SharePoint-Serverobjektmodell (siehe Kapitel 3), um das neue Element in die Liste einzufügen. Listing 6.6 zeigt die vollständige Implementierung des Webparts.

Listing 6.6 Vollständige Implementierung des Webparts *InsertRequestForContactWebPart*

```
namespace DevLeap.SP2010.WebParts.InsertRequestForContactWebPart {
    [ToolboxItemAttribute(false)]
    public class InsertRequestForContactWebPart : WebPart {
        protected TextBox RequesterFullName;
        protected TextBox RequesterEMail;
        protected TextBox Reason;
        protected Button SubmitRequestForContact;
        protected Label ErrorMessage;

        protected override void CreateChildControls() {
            this.RequesterFullName = new TextBox();
            this.RequesterFullName.Columns = 100;
            this.RequesterFullName.MaxLength = 255;
            this.Controls.Add(new LiteralControl("<div>Requester Full Name: "));
            this.Controls.Add(this.RequesterFullName);
            this.Controls.Add(new LiteralControl("</div>"));

            this.RequesterEMail = new TextBox();
            this.RequesterEMail.Columns = 100;
            this.RequesterEMail.MaxLength = 100;
            // "E-Mail des Anfragers"
            this.Controls.Add(new LiteralControl("<div>Requester EMail: "));
            this.Controls.Add(this.RequesterEMail);
            this.Controls.Add(new LiteralControl("</div>"));

            this.Reason = new TextBox();
            this.Reason.Columns = 100;
            this.Reason.MaxLength = 255;
            // "Grund"
            this.Controls.Add(new LiteralControl("<div>Reason: "));
            this.Controls.Add(this.Reason);
            this.Controls.Add(new LiteralControl("</div>"));

            this.SubmitRequestForContact = new Button();
            // "Kontaktanfrage absenden"
            this.SubmitRequestForContact.Text = "Submit Request for Contact";
            this.Controls.Add(new LiteralControl("<div>"));
            this.Controls.Add(this.SubmitRequestForContact);
            this.SubmitRequestForContact.Click +=
                new EventHandler(SubmitRequestForContact_Click);
            this.Controls.Add(new LiteralControl("</div>"));

            this.ErrorMessage = new Label();
            this.ErrorMessage.ForeColor = System.Drawing.Color.Red;
            this.Controls.Add(new LiteralControl("<div>"));
            this.Controls.Add(this.ErrorMessage);
            this.Controls.Add(new LiteralControl("</div>"));
        }
```

```
        void SubmitRequestForContact_Click(object sender, EventArgs e) {
            SPWeb web = SPControl.GetContextWeb(HttpContext.Current);

            try {
                SPList targetList = web.Lists["Requests for Contacts"];
                SPListItem newItem = targetList.Items.Add();
                newItem["Reason"] = this.Reason .Text;
                newItem["Requester full name"] = this.RequesterFullName.Text;
                newItem["Requester email"] = this.RequesterEMail.Text;
                newItem.Update();
            }
            catch (IndexOutOfRangeException) {
                this.ErrorMessage.Text =
                   "Kann Liste \"Requests for Contacts\" nicht finden";
            }
        }
    }
}
```

Der in Listing 6.6 hervorgehobene Code deklariert die *protected*-Variablen, in denen die ASP.NET-Serversteuerelemente gespeichert werden. In der Überschreibung der Methode *CreateChildControls* sehen Sie, wie Instanzen dieser Steuerelemente angelegt werden. Wichtig ist die Bindung zwischen dem serverseitigen *Click*-Ereignis der Schaltfläche *SubmitRequestForContact* und der Methode *SubmitRequestForContact_Click*. In diesem Ereignishandler legen Sie eine neue Instanz von *SPListItem* an, die für eine einzelne Kontaktanfrage steht. Dann stellen Sie die Felder dieses Elements zusammen und übergeben es schließlich an die *SPList*-Instanz.

Anhand dieses zweiten Beispiels können Sie sich vorstellen, dass Sie praktisch beliebige Webparts entwickeln können, indem Sie etwas ASP.NET-Code, benutzerdefinierte Steuerelemente und ein paar Zeilen .NET-Code kombinieren. Beispielsweise können Sie ein Webpart entwickeln, mit dem Endbenutzer auf eine Backend-Datenbank zugreifen, oder eines, das mit dem externen SOAP-Dienst eines anderen Herstellers kommuniziert. Vergessen Sie aber nicht, dass SharePoint eine robuste und sichere Umgebung ist und daher alle Anpassungen oder Lösungen genehmigt und autorisiert werden müssen, damit sie einwandfrei funktionieren. In Kapitel 7 gehe ich im Abschnitt »Bereitstellung, Sicherheit und Versionsverwaltung« darauf ein, welche Sicherheitsaspekte Sie beim Entwickeln und Bereitstellen benutzerdefinierter SharePoint-Webparts beachten müssen, damit sie sich in die Sicherheitsinfrastruktur von SharePoint einfügen.

Visuelle Webparts

In Listing 6.6 habe ich alle ASP.NET-Serversteuerelemente, aus denen sich das Webpart zusammensetzt, mit benutzerdefiniertem .NET-Code definiert. Es ist aber manchmal recht mühsam, ein Webpart vollständig im Code zu entwickeln, weil Sie oft Benutzeroberflächenattribute wie CSS-Stile, Steuerelementanordnung und -ausrichtung festlegen müssen. Außerdem gibt es Fälle, in denen Sie viele Steuerelemente in einem einzigen Webpart erstellen müssen; den gesamten Code dafür zu schreiben und zu pflegen ist eine schwierige Aufgabe. Eine mögliche Lösung, die vor SharePoint 2010 angewendet wurde, besteht darin, ein benutzerdefiniertes ASCX-Steuerelement zu definieren und es mit der Methode *LoadControl* der ASP.NET-Infrastruktur dynamisch in ein Webpart zu laden.

Seit SharePoint 2010 und Visual Studio 2010 gibt es nun eine einfache Lösung für dieses Problem. In Visual Studio 2010 steht eine Elementvorlage namens *Visuelles Webpart* (visual web part) zur Verfügung. Sie definiert ein Webpart, das ein benutzerdefiniertes ASCX-Steuerelement lädt. Ein solches Webpart tut genau das, was viele Entwickler in älteren SharePoint-Versionen von Hand erledigen mussten: dynamisch die Methode *LoadControl* eines externen ASCX-Steuerelements aufrufen. Listing 6.7 zeigt die Kernimplementierung eines visuellen Webparts namens *VisualInsertRequestForContactWebPart*.

Listing 6.7 Grundlegende Implementierung des Webparts *VisualInsertRequestForContactWebPart*

```
namespace DevLeap.SP2010.WebParts.VisualInsertRequestForContactWebPart {
    [ToolboxItemAttribute(false)]
    public class VisualInsertRequestForContactWebPart : WebPart {
        // Visual Studio kann diesen Pfad automatisch aktualisieren,
        // wenn Sie das Projektelement des visuellen Webparts ändern.
        private const string _ascxPath =
           @"~/_CONTROLTEMPLATES/DevLeap.SP2010.WebParts/
           VisualInsertRequestForContactWebPart/
           VisualInsertRequestForContactWebPartUserControl.ascx";

        protected override void CreateChildControls() {
            Control control = Page.LoadControl(_ascxPath);
            Controls.Add(control);
        }
    }
}
```

Neben dem hervorgehobenen Code, in dem das ASCX-Steuerelement dynamisch geladen wird, fügen Sie Ereignishandler und benutzerdefinierte Prozeduren in den Quellcode des Webparts ein. Die übrigen Einstellungen zur Struktur der Steuerelemente bleiben in der ASCX-Datei, die Sie in Listing 6.8 sehen.

Listing 6.8 Die ASCX-Datei für das Webpart *VisualInsertRequestForContactWebPart*

```
<%@ Assembly Name="$SharePoint.Project.AssemblyFullName$" %>
<%@ Assembly Name="Microsoft.Web.CommandUI, Version=14.0.0.0, Culture=neutral,
  PublicKeyToken=71e9bce111e9429c" %>
<%@ Register Tagprefix="SharePoint" Namespace="Microsoft.SharePoint.WebControls"
  Assembly="Microsoft.SharePoint, Version=14.0.0.0, Culture=neutral,
  PublicKeyToken=71e9bce111e9429c" %>
<%@ Register Tagprefix="Utilities" Namespace="Microsoft.SharePoint.Utilities"
  Assembly="Microsoft.SharePoint, Version=14.0.0.0, Culture=neutral,
  PublicKeyToken=71e9bce111e9429c" %>
<%@ Register Tagprefix="asp" Namespace="System.Web.UI"
  Assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
  PublicKeyToken=31bf3856ad364e35" %>
<%@ Import Namespace="Microsoft.SharePoint" %>
<%@ Register Tagprefix="WebPartPages" Namespace="Microsoft.SharePoint.WebPartPages"
  Assembly="Microsoft.SharePoint, Version=14.0.0.0, Culture=neutral,
  PublicKeyToken=71e9bce111e9429c" %>
```

```
<%@ Control Language="C#" AutoEventWireup="true"
  CodeBehind="VisualInsertRequestForContactWebPartUserControl.ascx.cs"
  Inherits="DevLeap.SP2010.WebParts.VisualInsertRequestForContactWebPart.
    VisualInsertRequestForContactWebPartUserControl" %>
<p>
    Requester full name:
    <asp:TextBox ID="RequesterFullName" runat="server" Columns="100"
        MaxLength="255"></asp:TextBox>
</p>
<p>
    Requester email:
    <asp:TextBox ID="RequesterEMail" runat="server" Columns="100"
        MaxLength="100"></asp:TextBox>
</p>
<p>
    Reason:
    <asp:TextBox ID="Reason" runat="server" Columns="100"
        MaxLength="255"></asp:TextBox>
</p>
<asp:Button ID="SubmitRequestForContact" runat="server"
    onclick="SubmitRequestForContact_Click" Text="Submit Request for Contact" />
<br /><br />
<asp:Label ID="ErrorMessage" runat="server" ForeColor="Red" Visible="False" />
```

Natürlich hat eine ASCX-Datei gegenüber normalem .NET-Code den Vorteil, dass Sie den ASCX-Code im Visual Studio 2010-Designer entwerfen können (Abbildung 6.6).

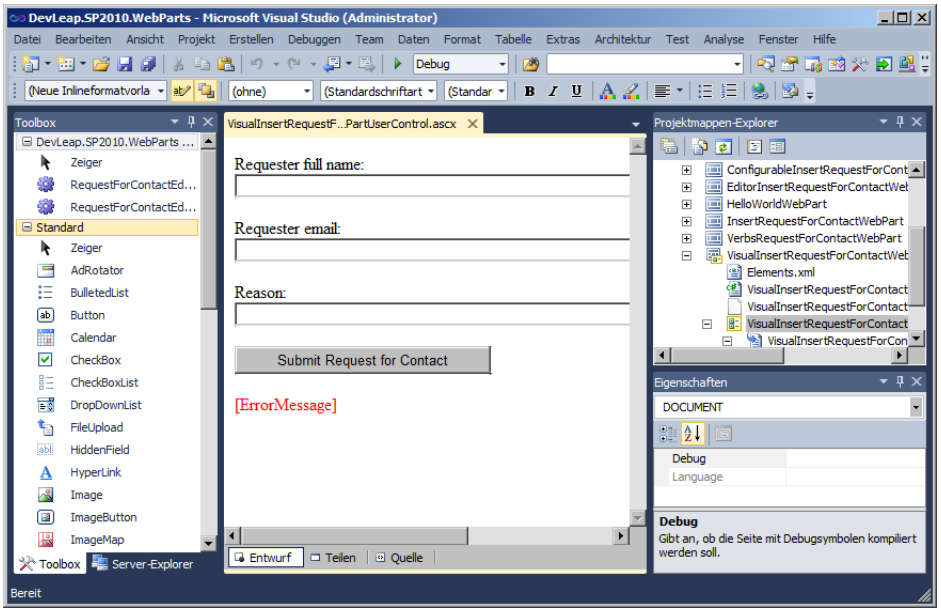

Abbildung 6.6 Der visuelle Designer der ASCX-Datei für *VisualInsertRequestForContactWebPart* in Visual Studio 2010

Auf den ersten Blick mag es aussehen, als wären visuelle Webparts die bessere und einfachere Lösung, besonders im Vergleich mit Standardwebparts. Es gibt aber einige Nebenwirkungen und Einschränkungen zu beachten, wenn Sie ein visuelles Webpart einsetzen. Zum Beispiel kann ein visuelles Webpart nicht als Sandkastenlösung bereitgestellt werden (mehr dazu in Kapitel 7), wodurch diese Lösung unter Umständen nicht so sicher ist, wie gefordert wird. Außerdem setzt ein visuelles Webpart voraus, dass die ASCX-Datei im freigegebenen Ordner *<SharePoint14_Root>\TEMPLATE\CONTROLTEMPLATES* bereitgestellt wird. Die Layoutelemente stehen somit allen Websites einer Farm zur Verfügung. Es gibt Fälle, in denen es flexibler ist, das Layout eines Webparts für jede Website beziehungsweise jeden Kunden individuell über XSLT verwalten und anpassen zu lassen. In Kapitel 7 gehe ich im Abschnitt »XSLT-Rendering« genauer auf dieses Thema ein.

HINWEIS Die Bezeichnung *<SharePoint14_Root>* ist der SharePoint-Stammordner, normalerweise *C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14*.

Konfigurierbare Webparts

In den letzten Beispielen haben Sie neue Elemente in eine vordefinierte Zielliste eingefügt. In SharePoint-Unternehmenslösungen können Webparts dagegen von autorisierten Benutzern konfiguriert werden. In diesem Abschnitt erfahren Sie, wie Sie konfigurierbare Webparts entwickeln und eine benutzerfreundliche Oberfläche für ihre Konfiguration bereitstellen.

Konfigurierbare Parameter

Wenn Sie konfigurierbare Webparts erstellen, besteht der erste Schritt darin festzulegen, welche Eigenschaften geändert werden können. Dazu brauchen Sie lediglich in der Klassendefinition des Webparts eine öffentliche Eigenschaft zu deklarieren und sie mit dem Attribut *WebBrowsableAttribute* sowie optional dem Attribut *PersonalizableAttribute* zu versehen. Listing 6.9 zeigt ein Webpart, das eine konfigurierbare Eigenschaft deklariert.

Listing 6.9 Ein Webpart, das eine konfigurierbare Eigenschaft zur Verfügung stellt

```
namespace DevLeap.SP2010.WebParts.ConfigurableInsertRequestForContactWebPart {
    [ToolboxItemAttribute(false)]
    public class ConfigurableInsertRequestForContactWebPart : WebPart {

        [WebBrowsable(true)]
        [Personalizable(PersonalizationScope.Shared)]
        public String TargetListTitle { get; set; }

        //
        // Code für CreateChildControls weggelassen ...
        //

        void SubmitRequestForContact_Click(object sender, EventArgs e) {
            SPWeb web = SPControl.GetContextWeb(HttpContext.Current);
```

```
            try {
                SPList targetList = web.Lists[this.TargetListTitle];

                SPListItem newItem = targetList.Items.Add();
                newItem["Reason"] = this.Reason.Text;
                newItem["Requester full name"] = this.RequesterFullName.Text;
                newItem["Requester email"] = this.RequesterEMail.Text;
                newItem.Update();
            }
            catch (IndexOutOfRangeException) {
                this.ErrorMessage.Text =
                   "Kann Liste \"Requests for Contacts\" nicht finden";
            }
        }
    }
}
```

Die Klasse *WebBrowsableAttribute* weist die Webpartsinfrastruktur an, die Eigenschaft im Konfigurationsfenster des Webparts verfügbar zu machen. Dieses Attribut hat den Parameter *Browsable* vom Typ *boolean*, in dem der Wert *true* übergeben wird, wenn Sie das Attribut über seinen Standardkonstruktor deklarieren. *PersonalizableAttribute* legt fest, dass die Eigenschaft angepasst werden kann, und legt den Umfang der Konfigurationsmöglichkeiten fest. Sie übergeben ihm entweder den Gültigkeitsbereich *User*, wenn die Eigenschaft für jeden Benutzer individuell angepasst werden kann, oder *Shared*, wenn die Änderungen an der Eigenschaft für alle Benutzer gelten.

Es gibt noch einige weitere Attribute, mit denen Sie die konfigurierbare Eigenschaft detaillierter definieren und so die Benutzerfreundlichkeit verbessern können. Zum Beispiel können Sie eine benutzerdefinierte Kategorie für die Eigenschaft definieren, indem Sie sie mit dem Attribut *CategoryAttribute* versehen. Die Bezeichnung der Eigenschaft legen Sie mit dem Attribut *WebDisplayNameAttribute* fest, und den Text des Tooltips, das der Endbenutzer angezeigt bekommt, steuern Sie mit dem Attribut *WebDescriptionAttribute*. Mit dem Attribut *DefaultValueAttribute* können Sie einen Standardwert für die Eigenschaft festlegen. Listing 6.10 zeigt eine vollständige Definition für die Eigenschaft *TargetListTitle*.

Listing 6.10 Ein Webpart, das eine konfigurierbare Eigenschaft mit allen nützlichen Attributen zur Verfügung stellt

```
[WebBrowsable(true)]
[Personalizable(PersonalizationScope.Shared)]
[WebDescription("Title of the Target list")]
[WebDisplayName("Target list")]
[Category("Data Foundation")]
public String TargetListTitle { get; set; }
```

Abbildung 6.7 zeigt die Benutzeroberfläche, die der Benutzer für eine konfigurierbare Eigenschaft angezeigt bekommt.

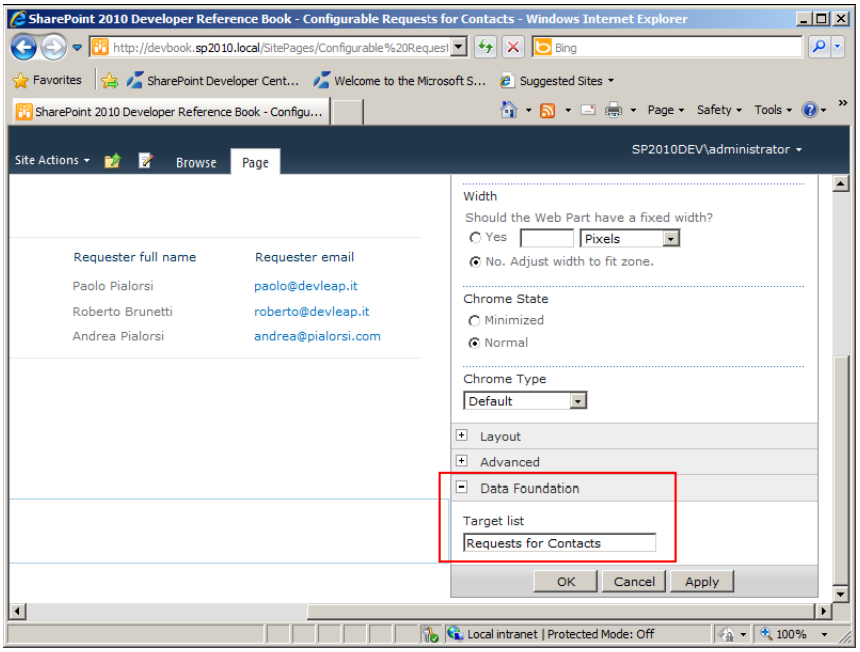

Abbildung 6.7 Der Konfigurationsabschnitt des Beispielwebparts

Der Editorbereich für das Webpart in Abbildung 6.7 wird von der SharePoint-Infrastruktur zur Verfügung gestellt. Er ist mit einigen SharePoint-spezifischen Klassen implementiert, den sogenannten Toolparts, die Sie mit eigenem Code anpassen können. Standardmäßig stellt SharePoint die Klasse *WebPartToolPart*, die die Benutzeroberfläche zum Bearbeiten der Standardeigenschaften eines Webparts (Titel, Rahmentyp, Größe und so weiter) zur Verfügung stellt, und die Klasse *CustomPropertyToolPart* bereit, die es automatisch ermöglicht, benutzerdefinierte Eigenschaften zu editieren.

Tabelle 6.2 schlüsselt auf, wie sich die Klasse *CustomPropertyToolPart* normalerweise verhält, wenn sie benutzerdefinierte Eigenschaften darstellt.

Tabelle 6.2 Standardverhalten der Klasse *CustomPropertyToolPart* beim Darstellen von benutzerdefinierten Eigenschaften

Typ der benutzerdefinierten Eigenschaft	Verhalten
Boolean	Zeigt ein Kontrollkästchen an.
Enum	Zeigt eine Dropdownliste an.
Integer	Zeigt ein Textfeld an.
String	Zeigt ein Textfeld an.
DateTime	Zeigt ein Textfeld an.

Benutzerdefinierte Toolpartklassen implementieren Sie, indem Sie eine Klasse von der abstrakten Basisklasse *ToolPart* aus dem Namespace *Microsoft.SharePoint.WebPartPages* ableiten. Damit ein benutzerdefiniertes Toolpart in SharePoint verfügbar ist, müssen Sie Ihre Webpartklasse allerdings von der ab-

strakten Basisklasse *Microsoft.SharePoint.WebPartPages.WebPart* ableiten, die von SharePoint zur Verfügung gestellt wird, und nicht von der abstrakten ASP.NET-Basisklasse *System.Web.UI.WebControls.WebParts.WebPart*. Überschreiben Sie dazu die Methode *GetToolParts* und geben Sie eine Auflistung der Toolparts zurück. Eine solche Anpassung funktioniert wegen der Abhängigkeit von der Assembly *Microsoft.SharePoint.dll* nur in SharePoint. Es wird aber nicht empfohlen, ein Webpart von *Microsoft.SharePoint.WebPartPages.WebPart* abzuleiten. Sie sollten immer ASP.NET-Webparts implementieren, die von *System.Web.UI.WebControls.WebParts.WebPart* abgeleitet sind, sofern Sie nicht unbedingt eine der wenigen Funktionen brauchen, die nur in SharePoint-Webparts zur Verfügung stehen. Auf dieses Thema konzentriert sich der Abschnitt »Die SharePoint-spezifische Klasse *WebPart*« am Ende dieses Kapitels.

EditorParts

Listing 6.10 definiert eine Eigenschaft, die erfordert, dass der Endbenutzer das Webpart von Hand konfiguriert, indem er den Namen der Zielliste eintippt. Sie haben das Standardverhalten von SharePoint und des vordefinierten *CustomPropertyToolPart* bereits genutzt. Aber obwohl es natürlich möglich ist, ein solches Webpart zu veröffentlichen, ist es sicherlich kein benutzerfreundlicher und fehlertoleranter Ansatz. Eine bessere Lösung wäre, eine Dropdownliste mit allen Listen anzuzeigen, die in der aktuellen Website vorhanden sind. Auf diese Weise werden Tippfehler und zeitaufwendiges Debuggen vermieden. Um die Benutzeroberfläche für die Konfiguration von Webparts anzupassen, erstellen Sie benutzerdefinierte Klassen, die sogenannten EditorParts aus der Webpartinfrastruktur von ASP.NET. EditorParts sind Steuerelemente, die in einer bestimmten *WebPartZone*, der *EditorZone* gehostet werden. Sie sind Standardwebparts sehr ähnlich, sind allerdings nicht von der Klasse *WebPart*, sondern von der Basisklasse *EditorPart* abgeleitet. Diese Basisklasse verknüpft das EditorPart mit dem momentan bearbeiteten Webpart. Um ein Webpart mit einem benutzerdefinierten EditorPart bereitzustellen, müssen Sie die Implementierung der Schnittstelle *IWebEditable* aus der Basisklasse des Webparts überschreiben. Listing 6.11 zeigt die Definition dieser Schnittstelle.

Listing 6.11 Die Definition der Schnittstelle *IWebEditable*

```
public interface IWebEditable {
    EditorPartCollection CreateEditorParts();
    object WebBrowsableObject { get; }
}
```

Die Schnittstelle deklariert die Methode *CreateEditorParts*, die eine Auflistung mit EditorParts zurückgibt, mit denen die Webparts ergänzt werden. Außerdem definiert die Schnittstelle eine öffentliche schreibgeschützte Eigenschaft, die einen Verweis auf das konfigurierbare Objekt liefert, das von den EditorParts bearbeitet wird. Üblicherweise gibt die Eigenschaft *WebBrowsableObject* die aktuelle Webpartinstanz (*this*) zurück. Listing 6.12 enthält die neue Implementierung des benutzerdefinierten Webparts.

Listing 6.12 Das neue benutzerdefinierte Webpart implementiert die Schnittstelle *IWebEditable*

```
namespace DevLeap.SP2010.WebParts.EditorInsertRequestForContactWebPart {
    [ToolboxItemAttribute(false)]
    public class EditorInsertRequestForContactWebPart : WebPart {

        [WebBrowsable(false)]
        [Personalizable(PersonalizationScope.Shared)]
        public Guid TargetListID { get; set; }

        //
        // Code für CreateChildControls weggelassen ...
        //

        void SubmitRequestForContact_Click(object sender, EventArgs e) {
            SPWeb web = SPControl.GetContextWeb(HttpContext.Current);

            try {
                SPList targetList = web.Lists[this.TargetListID];
                SPListItem newItem = targetList.Items.Add();
                newItem["Reason"] = this.Reason.Text;
                newItem["Requester full name"] = this.RequesterFullName.Text;
                newItem["Requester email"] = this.RequesterEMail.Text;
                newItem.Update();
            }
            catch (IndexOutOfRangeException) {
                this.ErrorMessage.Text =
                    "Kann Liste \"Requests for Contacts\" nicht finden";
            }
        }

        public override EditorPartCollection CreateEditorParts() {
            RequestForContactEditorPart editorPart =
                new RequestForContactEditorPart();
            editorPart.ID = this.ID + "_RequestForContactEditorPart";

            EditorPartCollection editorParts =
              new EditorPartCollection(base.CreateEditorParts(),
              new EditorPart[] { editorPart });
            return editorParts;
        }

        public override object WebBrowsableObject {
            get { return(this); }
        }
    }
}
```

In Listing 6.12 habe ich die *String*-Eigenschaft *TargetListTitle* durch die Eigenschaft *TargetListID* vom Typ *Guid* ersetzt, damit sie die eindeutige ID der Zielliste aufnimmt. Anhand dieser ID suche ich im Ereignishandler *SubmitRequestForContact_Click* nach der Listeninstanz. Das Attribut *WebBrowsable* der Eigenschaft habe ich deaktiviert, damit sie nicht in der Standardeigenschaftstabelle des Webpart-Editors angezeigt wird. Diese Eigenschaft wird mit dem benutzerdefinierten EditorPart verwaltet.

HINWEIS Wenn Sie das Attribut *WebBrowsable* einer Eigenschaft, die auch über ein benutzerdefiniertes EditorPart konfiguriert werden kann, nicht deaktivieren, können Ihre Endbenutzer die Eigenschaft sowohl im benutzerdefinierten EditorPart als auch in der Standardeigenschaftstabelle bearbeiten, die von der SharePoint-Klasse *CustomPropertyToolPart* zur Verfügung gestellt wird. Das ist natürlich verwirrend für die Endbenutzer und sollte vermieden werden.

Anschließend habe ich die Methode *CreateEditorParts* so überschrieben, dass sie die Implementierung der Basisklassenmethode aufruft und ein benutzerdefiniertes EditorPart namens *RequestForContactEditorPart* zur Auflistung der verfügbaren EditorParts für das aktuelle Webpart hinzufügt. Schließlich habe ich eine benutzerdefinierte ID für die EditorPart-Instanz definiert, die sich aus der eindeutigen ID des aktuellen Webparts ableitet. So wird auch die EditorPart-ID eindeutig.

EditorPart ist eine abstrakte Basisklasse. Sie stellt einige virtuelle oder abstrakte Methoden und Eigenschaften bereit, die nützlich sind, um die Bearbeitung des Zielwebparts zu verwalten. Zum Beispiel haben alle Klassen, die von *EditorPart* abgeleitet sind, die Eigenschaft *WebPartToEdit*. Sie verweist auf die Webpartinstanz, die das EditorPart momentan bearbeitet. Mit den abstrakten Methoden *ApplyChanges* und *SyncChanges* sichern Sie Änderungen am bearbeiteten Webpart beziehungsweise laden seine aktuelle Konfiguration.

Listing 6.13 zeigt die Implementierung der Klasse *RequestForContactEditorPart*.

Listing 6.13 Die Implementierung der Klasse *RequestForContactEditorPart*

```
public class RequestForContactEditorPart : EditorPart {
    protected DropDownList targetLists;

    protected override void CreateChildControls() {
        this.targetLists = new DropDownList();

        SPWeb web = SPControl.GetContextWeb(HttpContext.Current);
        foreach (SPList list in web.Lists) {
            this.targetLists.Items.Add(new ListItem(list.Title, list.ID.ToString()));
        }

        this.Title = "Request for Contact EditorPart";
        // "Zielliste auswählen"
        this.Controls.Add(new LiteralControl("Select the target List:<br>"));
        this.Controls.Add(this.targetLists);
        this.Controls.Add(new LiteralControl("<br> <br>"));
    }

    public override bool ApplyChanges() {
        EnsureChildControls();
```

```
        EditorInsertRequestForContactWebPart wp =
           this.WebPartToEdit as EditorInsertRequestForContactWebPart;
        if (wp != null) {
            wp.TargetListID = new Guid(this.targetLists.SelectedValue);
        }
        return (true);
    }

    public override void SyncChanges() {
        EnsureChildControls();

        EditorInsertRequestForContactWebPart wp =
           this.WebPartToEdit as EditorInsertRequestForContactWebPart;
        if (wp != null) {
            ListItem selectedItem =
               this.targetLists.Items.FindByValue(wp.TargetListID.ToString());
            if (selectedItem != null) {
                this.targetLists.ClearSelection();
                selectedItem.Selected = true;
            }
        }
    }
}
```

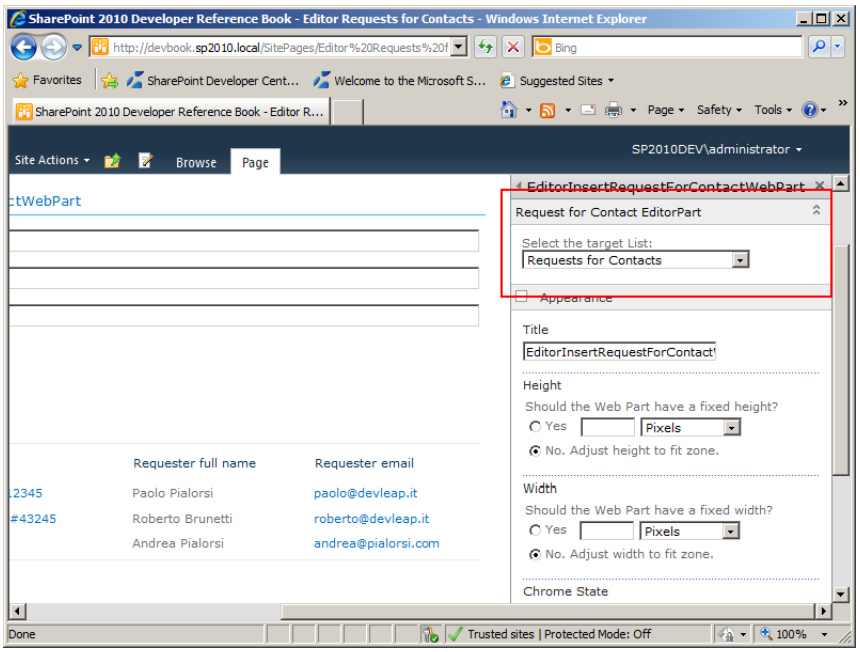

Abbildung 6.8 Der Konfigurationsabschnitt des Beispielwebparts, nachdem ein EditorPart erstellt wurde

Wie jedes andere Webpart muss auch ein EditorPart die Hierarchie seiner untergeordneten Steuerelemente anlegen, um seinen Inhalt anzuzeigen. In Listing 6.13 erstelle ich in der Überschreibung der Methode *CreateChildControls* eine Dropdownliste und binde sie an die Auflistung der Listen in der aktuellen Website.

Dann speichere ich in der Überschreibung der Methode *ApplyChanges* die momentan ausgewählte Listen-ID in der Eigenschaft *TargetListID* der aktuellen Webpartinstanz. Und in der Methode *SyncChanges* wähle ich in der Dropdownliste automatisch den Eintrag mit der ID aus, die dem Wert der aktuellen Eigenschaft *TargetListID* entspricht. Abbildung 6.8 zeigt die Darstellung des benutzerdefinierten EditorParts.

Eine SharePoint-Unternehmenslösung stellt üblicherweise für alle ihre Webparts umfangreiche Konfigurationsparameter zur Verfügung, die über benutzerdefinierte EditorParts konfiguriert und angepasst werden können.

Verarbeiten der Anzeigemodi

Beim Entwickeln von Webparts ist es früher oder später nötig, die Darstellung eines benutzerdefinierten Webparts abhängig vom Status der Seite zu ändern, in die es eingebettet ist. Eine Seite, die Webparts hostet, kann im Anzeigemodus (display mode) dargestellt werden, wenn der Endbenutzer die Website ansieht, im Entwurfsmodus (design mode), wenn der Benutzer das Seitenlayout anpasst, oder im Bearbeitungsmodus (edit mode), wenn der Endbenutzer die Seite oder ihre Steuerelemente bearbeitet.

Um den Anzeigemodus einer Seite zu ermitteln und ein Webpart passend darzustellen, müssen Sie die Eigenschaft *DisplayMode* von *WebPartManager* (*SPWebPartManager* in SharePoint) auslesen. Listing 6.14 zeigt ein Beispielwebpart, das seine Darstellung an den aktuellen Wert von *DisplayMode* anpasst.

Listing 6.14 Ein Webpart zeigt seinen Inhalt relativ zum Anzeigemodus der aktuellen Seite an

```
protected override void CreateChildControls() {
    if (this.WebPartManager.DisplayMode == WebPartManager.BrowseDisplayMode) {
        // Anzeigemodus
        // Standardinhalt anzeigen
    }
    else if (this.WebPartManager.DisplayMode == WebPartManager.DesignDisplayMode) {
        // Entwurfsmodus
        // "Bitte schalten Sie in den Anzeigemodus, um dieses Webpart zu benutzen."
        this.Controls.Add(new LiteralControl("<div>
           Please move to Display mode to use this Web Part.</div>"));
    }
    else if (this.WebPartManager.DisplayMode == WebPartManager.EditDisplayMode) {
        // Bearbeitungsmodus
        // "Bitte schalten Sie in den Anzeigemodus, um dieses Webpart zu benutzen, oder
        //  konfigurieren Sie hier im Bearbeitungsmodus seine Eigenschaften."
        this.Controls.Add(new LiteralControl("<div>
           Please move to Display mode to use this Web Part or configure its
           properties, since you are in Edit mode.</div>"));
    }
}
```

Alle von *WebPart* abgeleiteten Klassen haben eine Eigenschaft, die auf die aktuelle *WebPartManager*-Instanz verweist. Über diese Eigenschaft haben Sie Zugriff auf *DisplayMode* und viele andere Kontexteigenschaften. Sie können auch *WebPartManager* nutzen, um Ereignisse zu abonnieren, die bei Änderungen an *DisplayMode* ausgelöst werden. Zum Beispiel gibt es die Ereignisse *DisplayModeChanging* und *DisplayModeChanged*, mit denen Sie den Status von *DisplayMode* überwachen können.

Benutzerdefinierte Webpartverben

Eine weitere nützliche Anpassungsmöglichkeit für Webparts ist die Definition benutzerdefinierter Webpartverben. Webpartverben sind Menüelemente, die im Kontextmenü eines Webparts angezeigt werden (Abbildung 6.9).

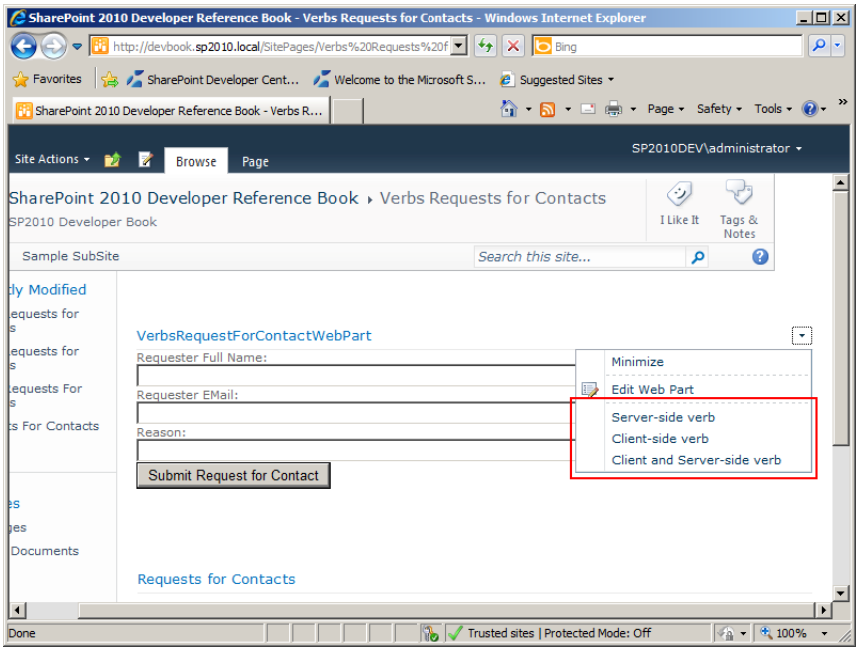

Abbildung 6.9 Benutzerdefinierte Verben in einem Webpart

Um benutzerdefinierte Verben zu konfigurieren, überschreiben Sie die schreibgeschützte Eigenschaft *Verbs* aus der Basisklasse *WebPart*. Diese Eigenschaft gibt eine Auflistung des Typs *WebPartVerbCollection* zurück und kann benutzt werden, um das Kontextmenü des Webparts völlig neu zu gestalten. Verben sind Objekte vom Typ *WebPartVerb*, es gibt drei unterschiedliche Arten:

- **Serverseitig** Verben, die ein POST brauchen, um ihre Aufgabe zu erfüllen. Sie arbeiten auf der Serverseite.
- **Clientseitig** Verben, die ihre Aufgabe mit JavaScript ausführen. Sie arbeiten auf der Clientseite.
- **Client- und serverseitig** Verben, die zuerst clientseitiges JavaScript ausführen, gefolgt von serverseitigem Code, sofern der clientseitige Code die Anforderung nicht abbricht.

Listing 6.15 zeigt einen Ausschnitt aus dem Beispielwebpart, das all drei Arten benutzerdefinierter Verben unterstützt.

Listing 6.15 Ein Webpart mit benutzerdefinierten Verben

```
public override WebPartVerbCollection Verbs {
    get {
        WebPartVerb serverSideVerb = new WebPartVerb("serverSiteVerbId",
           handleServerSideVerb);
        serverSideVerb.Text = "Server-side verb";
        // "Clientseitiges Verb ausgewählt"
        WebPartVerb clientSideVerb = new WebPartVerb("clientSideVerbId",
           "javascript:alert('Client-side Verb selected');");
        clientSideVerb.Text = "Client-side verb";

        WebPartVerb clientAndServerSideVerb = new
            WebPartVerb("clientAndServerSideVerbId",
            handleServerSideVerb, "javascript:alert('Client-side Verb selected');");
        // "Client- und serverseitiges Verb"
        clientAndServerSideVerb.Text = "Client and Server-side verb";

        WebPartVerbCollection newVerbs = new WebPartVerbCollection(
            new WebPartVerb[] {
                serverSideVerb, clientSideVerb, clientAndServerSideVerb,
            }
            );
        return (new WebPartVerbCollection(base.Verbs, newVerbs));
    }
}

protected void handleServerSideVerb(Object source, WebPartEventArgs args) {
    EnsureChildControls();

    // "Sie haben ein serverseitiges Ereignis ausgelöst!"
    this.GenericMessage.Text = "You raised a server-side event!";
}
```

Dieser Beispielcode enthält die Implementierung der Eigenschaft *Verbs*, in der die Verben von Hand definiert und konfiguriert werden. Danach werden sie zur Auflistung der Webpartverben hinzugefügt.

Im Allgemeinen werden benutzerdefinierte Verben in Intranet- oder Extranetlösungen definiert, um besondere Funktionen wie das Aktualisieren des Inhalts oder das Öffnen eines Popupfensters zu unterstützen. In CMS-Lösungen werden sie dagegen nur selten eingesetzt, weil das Kontextmenü darin normalerweise deaktiviert ist.

Die SharePoint-spezifische Klasse *WebPart*

Wie Sie am Anfang dieses Kapitels erfahren haben, besteht in SharePoint die Möglichkeit, Webparts von einer SharePoint-spezifischen Basisklasse abzuleiten, statt die ASP.NET-Standardklasse zu verwenden. Die so entstandenen Webparts sind nahtlos in die ASP.NET-Webpartsinfrastruktur integriert, weil die SharePoint-Klasse *WebPart* intern von ihrem ASP.NET-Gegenstück abgeleitet ist. Diese Webparts können aber nur in SharePoint benutzt werden. Sie bieten einige zusätzliche Funktionen, die es unter ganz bestimmten Umständen sinnvoll machen, solche SharePoint-spezifischen Webparts zu implementieren. Der zusätzliche Funktionsumfang bietet folgende Vorteile:

- **Unterstützung für SharePoint-Toolparts** Sie haben bereits gesehen, was ein Toolpart ist und was Sie damit machen können.
- **Ersatztoken für Pfad/Code** Sie können damit Token (Platzhalter) in den ausgelieferten HTML-Code eines SharePoint-Webparts einarbeiten. Die SharePoint-Infrastruktur ersetzt sie dann durch die tatsächlichen Werte. Es gibt Token für den aktuellen Benutzernamen, die LCID der Website und so weiter.
- **Seitenübergreifende Verbindungen und Verbindungen zwischen Webparts, die außerhalb einer Webpartzone liegen** Webparts können miteinander verknüpft werden, um Master-Detail-Lösungen zu entwickeln (siehe Kapitel 7). SharePoint-spezifische Webparts unterstützen seitenübergreifende Verbindungen, während ASP.NET-Standardwebparts nur Verbindungen innerhalb derselben Seite beherrschen. SharePoint-Webparts können auch miteinander verbunden werden, wenn sie sich außerhalb einer Webpartzone befinden.
- **Clientseitige Verbindungen** Dies sind Verbindungen zwischen SharePoint-spezifischen Webparts, die durch clientseitigen Code (JavaScript) implementiert werden.
- **Zwischenspeichern von Daten** Es gibt eine Infrastruktur für die Zwischenspeicherung von Daten, die es erlaubt, Webpartsdaten in der Inhaltsdatenbank abzulegen.

Zusammenfassung

In diesem Kapitel haben Sie erfahren, was Webparts sind, auf welcher Architektur sie aufbauen und wie sie so entwickelt und bereitgestellt werden, dass ihr Aussehen und Verhalten nach Belieben angepasst werden können. Insbesondere haben Sie gesehen, wie Sie konfigurierbare und anpassbare Webparts erstellen, die dem Endbenutzer eigene EditorParts, Toolparts und benutzerdefinierte Verben zur Verfügung stellen. In Kapitel 7 lernen Sie fortgeschrittene Techniken kennen, beispielsweise Webpartverbindungen, Unterstützung für AJAX und Silverlight, asynchrone Programmierung, XSLT-Rendering, Webpartsicherheit, Bereitstellung und Versionsverwaltung.