Kapitel 26

SharePoint-Programmierung

In diesem Kapitel:

In diesem Kapitel erwartet Sie eine Einführung in die SharePoint-Programmierung. Ein spannendes aber auch ein nicht ganz leichtes Thema, für das ein Grundverständnis für die Programmierung sicherlich vorteilhaft ist. Dennoch: Neueinsteigern raten wir, sich nicht gleich entmutigen zu lassen. Nehmen Sie sich Zeit und programmieren Sie die Beispiele einfach mal nach.

In den Beispielen dieses Kapitels demonstrieren wir den Zugriff auf Websitesammlungen, Websites, Listen, Listenelemente und Metadaten. Außerdem erfahren Sie, wie man grafische Benutzeroberflächen mit Steuerelementen programmieren kann.

Sie erhalten damit einen ersten Einblick in die Entwicklung unter SharePoint 2010 und wie bereits erwähnt, können Sie die Anwendungen auch nachprogrammieren. Wir zeigen Ihnen wie Sie

- Listen anzeigen
- Listen durchsuchen
- Listenelemente ändern
- Listenelemente löschen
- Webparts programmieren

Sie erfahren, wie Sie über ein Netzwerk auf SharePoint programmatisch zugreifen können und Sie sehen Beispiele für die Programmierung von Silverlight-Anwendungen für SharePoint.

Den Abschluss bildet eine Übersicht zum Thema *Sandboxed Solutions*. Die deutsche Bezeichnung hierfür ist Sandkastenlösungen. In diesem Abschnitt lesen Sie, wie Entwickler oder Poweruser Webparts installieren können, die auf Websitesammlungsebene bereitgestellt werden. Praktisch bedeutet dies, dass entsprechende Webparts von Websitesammlungsadministratoren installiert werden können. Administratorrechte auf Farmebene sind hierfür nicht notwendig.

HINWEIS In diesem Kapitel wird auf einer englischen Entwicklungsumgebung gearbeitet. Englische Menüoptionen und Bezeichnungen der verwendeten Arbeitsbereiche sind an den Stellen übersetzt, wo die Zuordnung eventuell nicht eindeutig ist.

Definition der wichtigsten Begriffe

Da es sich in diesem Kapitel um ein technisches Kapitel handelt, ist es sinnvoll, zunächst einige Begriffe (Tabelle 26.1Tabelle 26.1) zu definieren. Dabei wird einen pragmatischer Ansatz verfolgt – die Definitionen werden vereinfacht ohne sie zu verfälschen.

Tabelle 26.1 Übersicht über die wichtigsten Begriffe im Bereich Programmierung

Begriff	Definition
Anwendungsfall	Der *Anwendungsfall* oder *Use Case* beschreibt ein von der Anwendung zu lösendes Problem. Die Beschreibung des Anwendungsfalls ist die Grundlage für dessen programmatische Umsetzung.
Klasse	Eine *Klasse* beschreibt ein Objekt, seine Eigenschaften und sein Verhalten. Das Verhalten wird in *Methoden* definiert. Häufig wird für Methode auch der Begriff *Funktion* verwendet.
Objekt	Ein *Objekt* ist die Instanz einer Klasse. Es gibt also nur eine Klasse, z.B. »SPList«, aber viele Objekte davon.

Tabelle 26.1 Übersicht über die wichtigsten Begriffe im Bereich Programmierung *(Fortsetzung)*

Begriff	Definition
Assembly	Eine *Assembly* ist die Zusammenfassung von Klassen in einer Datei. Assemblys, die ausführbare Klassen beinhalten, aus der Sicht des Anwenders also Programme, haben die Erweiterung .exe. Assemblys, die Klassen zur Verfügung stellen, die Programme nutzen, haben die Erweiterung .dll.
Framework	Ein *Framework* ist die Summe vieler Klassen, die einen bestimmten Anwendungsbereich abdecken. Das für unser Thema entscheidende Framework ist das .NET Framework, welches in verschiedenen Versionen vorliegt.
API	Der Begriff *API* (Application Programmable Interface) ist eine andere Bezeichnung für ein Framework. Der Begriff betont vor allem die Nutzung der Frameworkklassen für die Entwicklung eigener Anwendungen. APIs sind oft auch kommerzielle Frameworks von Drittanbieter mit dem Ziel, dem Entwickler die Arbeit zu erleichtern.
Runtime	Die *Runtime* oder *Laufzeitumgebung* liefert alle technischen Voraussetzungen, damit ein Programm ausgeführt werden kann. In diesem Sinne ist .NET auch eine Runtime. Eine andere Bezeichnung hierfür ist *Virtuelle Maschine*.
Client	Ein *Client* ist ein Programm, welches die Dienste anderer Programme, der Server oder Services, nutzt. Ein Server ist demnach ein Programm, das Dienste zur Verfügung stellt. Diesen Zusammenhang gibt es auch auf der Ebene der Betriebssysteme, die man im Clientbetriebssystem, z.B. Window 7 und Serverbetriebssystem, z.B. Windows Server 2008 R2, unterteilt. Demnach muss eine Serveranwendung auf einem Serverbetriebssystem laufen, eine Clientanwendung auf einem Clientbetriebssystem.
Projekt	Ein *Projekt* definiert den Rahmen einer Anwendungsentwicklung. Das Projekt besteht aus Programmdateien, Konfigurationsdateien, Referenzen auf andere Projekte, Assemblys und optional aus Ressourcen wie z.B. Multimediadateien.
Projekttemplates	*Projekttemplates* oder Projektvorlagen sind Zusammenfassungen von Projektdateien, die für einen bestimmten Anwendungstyp stehen. So gibt es z.B. ein Template für die Entwicklung von Webparts für SharePoint 2010.
Referenz	Eine *Referenz* ist der Verweis auf eine Datei in einem Projekt. Dadurch wird die Datei in das Projekt eingebunden. Referenzen dienen dazu, gleiche Dateien in unterschiedlichen Projekten zu verwenden.
Entwicklungsumgebung	Mit einer *Entwicklungsumgebung* bezeichnet man ein Programm, das zum Editieren, Testen und Kompilieren einer Anwendung verwendet werden kann. Entwicklungsumgebungen bieten eine Vielzahl von Funktionen. Visual Studio von Microsoft ist die Standardentwicklungsumgebung für .NET-Anwendungen und damit auch für die SharePoint-Programmierung.
Control	Ein *Control*, auch *Steuerelement* oder *Oberflächenkomponente* genannt, ist eine Klasse des .NET Framework, deren Objekte für die Programmierung von Benutzeroberflächen verwendet werden. Controls sind damit die Bausteine für die Programmierung von visuellen Webparts.
Debugging	Mit *Debugging* wird die systematische Fehlersuche während der Programmentwicklung bezeichnet. Gute Entwicklungswerkzeuge wie Visual Studio besitzen immer auch gute Debugging-Funktionalitäten.
Feature	Unter SharePoint werden Programmmodule, die in verschiedenen Anwendungen verwendet werden können, als *Features* bezeichnet.
Solution	Ein Programm wird in der SharePoint-Entwicklung als *Solution* bezeichnet.

HINWEIS Für Einsteiger kann die Versionierung von .NET verwirrend sein. Dies gilt insbesondere für die SharePoint-Programmierung. Obwohl die aktuelle Version von .NET die Version 4.0 ist, müssen SharePoint-Anwendungen mit der älteren Version 3.5 entwickelt werden. Ansonsten kommt es beim Ausführen von Programmen zu nicht nachvollziehbaren Fehlermeldungen.

Das .NET Framework bietet die Möglichkeit, Windows-Anwendungen zu schreiben ohne technische Kenntnisse von Windows als Betriebssystem zu haben. So erlaubt Ihnen das Framework z.B. beim Klick einer Schaltfläche programmatisch reagieren zu können. Sie müssen dabei das Klickereignis der Schaltfläche nicht selbst programmieren. Das Framework erkennt den Klick und leitet das Ergebnis an die .NET-Anwendung weiter. SharePoint-Programme werden mit .NET Framework entwickelt.

Damit ist die Übersicht zu den grundlegenden Begriffen für die SharePoint-Entwicklung abgeschlossen. Im weiteren Verlauf werden wir an entsprechenden Stellen nochmals etwas vertiefend auf den einen oder anderen Begriff eingehen.

Das Programmiermodell von SharePoint

Um die SharePoint-Programmierung wirklich zu verstehen, müssen Sie sich mit dem Programmiermodell auseinander setzen, also mit der Frage, wie man eigentlich für SharePoint programmiert. Dazu müssen Sie die Architektur von SharePoint Server sehr gut kennen.

Deshalb finden Sie nachfolgend eine kleine Liste von SharePoint-Komponenten bzw. -Kategorien. Diese Liste soll Ihnen als Orientierungshilfe dienen:

- Websitesammlung – Site Collection
- Website – Site
- Website höchster Ebene
- Website unterhalb einer Website – Unterwebsite
- Liste – Dokumentenbibliothek
- Metadaten
- Webpart
- Webpartseite

HINWEIS Wenn Sie in der Liste Begriffe finden, mit denen Sie nichts anfangen können, empfehlen wir Ihnen die entsprechenden Kapitel in diesem Buch nochmals zu lesen.

Für jede dieser Komponenten gibt es eine entsprechende Abbildung als SharePoint-Klasse. Vereinfacht ausgedrückt bedeutet dies, dass Sie z.B. auf eine Liste in einem Programm zugreifen können, indem Sie die Klasse verwenden, die eine SharePoint-Liste darstellt. Dabei müssen Sie aber auch wissen, dass Listen nur innerhalb einer Website existieren, das heißt in einer Websitesammlung existieren Sie nicht. Deshalb muss eine Websitesammlung mindestens eine Website haben – die Website höchster Ebene.

Und damit kommen wir dann auch zum Programmiermodell. Dieses umfasst:

- Visual Studio als Entwicklungswerkzeug
- .NET Framework

- Installierten SharePoint Server
- SharePoint-Klassen

Darüber hinaus legt das Programmiermodell fest, auf welchem Entwicklungssystem programmiert werden muss. In der Regel ist das eine Entwicklungsmaschine unter Windows Server 2008 R2 mit installiertem SharePoint 2010, also ein Serverbetriebssystem. Mit entsprechendem Know-how können Sie auch unter Windows 7, eigentlich einem Clientbetriebssystem, für SharePoint entwickeln. Allerdings müssen dazu einige Komponenten, die im Windows Server 2008 R2 schon enthalten sind, nachinstalliert werden. Abschließend müssen Sie noch festlegen, in welcher Programmiersprache Sie die Programme schreiben wollen. Dies wird aber im Wesentlichen davon abhängen, ob Sie Visual Basic oder C# beherrschen.

TIPP Die Beispiele in diesem Kapitel wurden mit C# entwickelt. Wenn Sie Kenntnisse in Visual Basic haben, dann dürfte die Umsetzung in Visual Basic kein Problem darstellen. Einsteiger sollten unserer Empfehlung nach direkt mit C# beginnen.

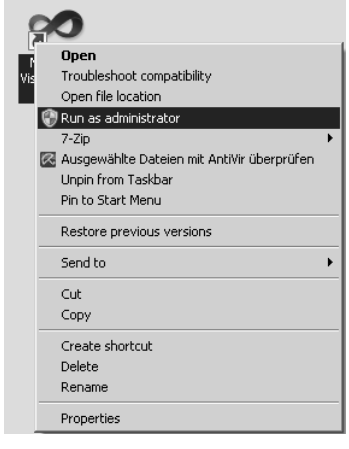
Bevor Sie also jetzt in die Umsetzung der nachfolgend beschriebenen Beispiele gehen, müssen Sie folgende Umgebung schaffen:

- Windows Server 2008 R2 oder entsprechend konfiguriertes Windows 7
- SharePoint Server 2010
- Visual Studio 2010
- Eine SiteCollection mit einer Website höchster Ebene und einer Website in der Website höchster Ebene – also eine Website als Unterwebsite.

TIPP Entwickeln Sie immer als Administrator sowohl auf Betriebssystemebene als auch in SharePoint. Sollte dies nicht möglich sein, dann benötigt Ihr Benutzer-Konto zu mindest Administratorrechte. Starten Sie in diesem Fall die Entwicklungsumgebung immer auch mit Administratorrechten:

1. Führen Sie einen Rechtsklick auf das Programmicon aus
2. Wählen Sie im Kontextmenü die Option *Run as administrator* (Abbildg. 26.1Abbildung 26.1).

Abbildg. 26.1 Visual Studio im Administratormodus starten



Objektmodelle

Ein *Objektmodell* ist der abstrakte Begriff für die Summe aller Klassen bzw. Objekte, die für die Programmierung eines Anwendungsbereichs als API benutzt werden können. Mit SharePoint 2010 können Entwickler jetzt auf zwei Objektmodelle zugreifen:

- Server-Objektmodell
- Client-Objektmodell

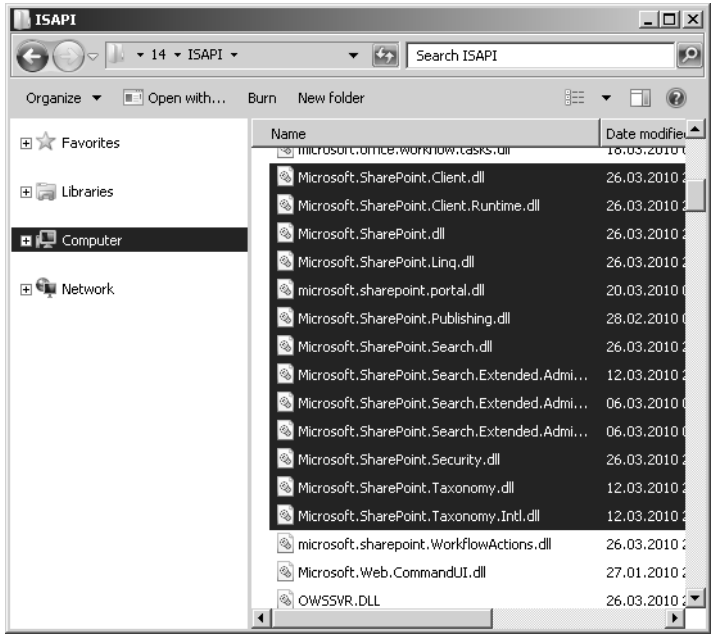
Für unser Themengebiet sind das alle Klassen, die Microsoft für die SharePoint-Programmierung zur Verfügung stellt.

WICHTIG Diese Klassen sind in Assemblys zusammengefasst und liegen standardmäßig im Pfad *C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\ISAPI*.

HINWEIS Das Akronym ISAPI steht für **I**nternet **S**erver **API**. Alle Assemblys dieses Verzeichnisses bilden also die API für die Entwicklung von SharePoint-Solutions. Die ISAPI umfasst dabei nicht nur SharePoint-Klassen.

In Abbildung Abbildg. 26.226.2 sehen Sie die Dateien markiert, die den Kern der SharePoint-API bilden.

Abbildg. 26.2 SharePoint-Assemblys

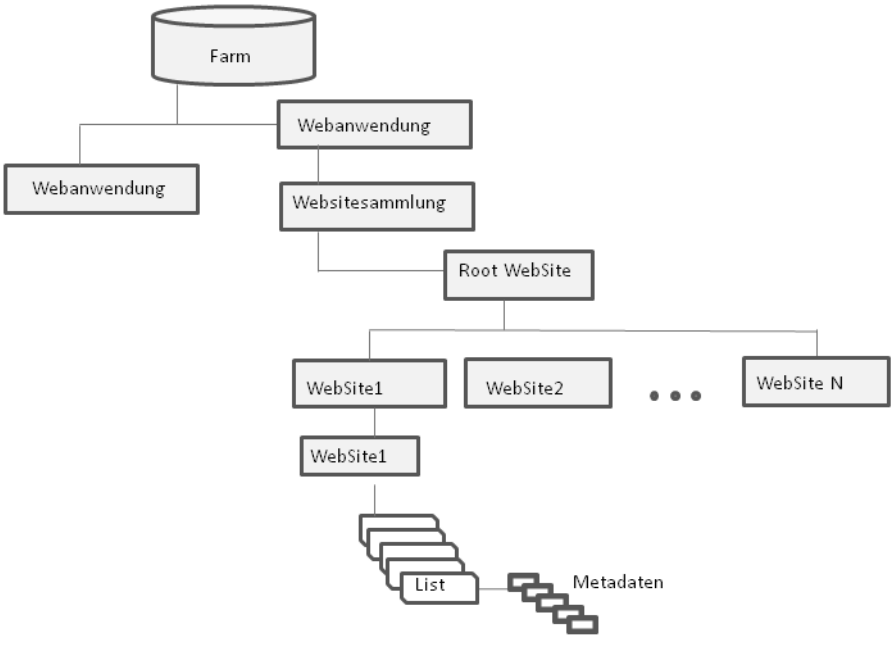


Das Objektmodell ist hierarchisch aufgebaut und bildet damit die SharePoint-Hierarchie ab. Aus programmatischer Sicht bedeutet dies:

- Der Entwickler hat über ein Site-Objekt, das für eine Websitesammlung oder SiteCollection steht, Zugriff auf alle Websites.
- Über eine Website hat er Zugriff auf alle Listen der Website.
- Über eine Liste erfolgt der Zugriff auf alle Metadaten oder Spalten der Liste.

Abbildung Abbildg. 26.326.3 verdeutlicht den Zusammenhang.

Abbildg. 26.3 Hierarchie des SharePoint-Objektmodells

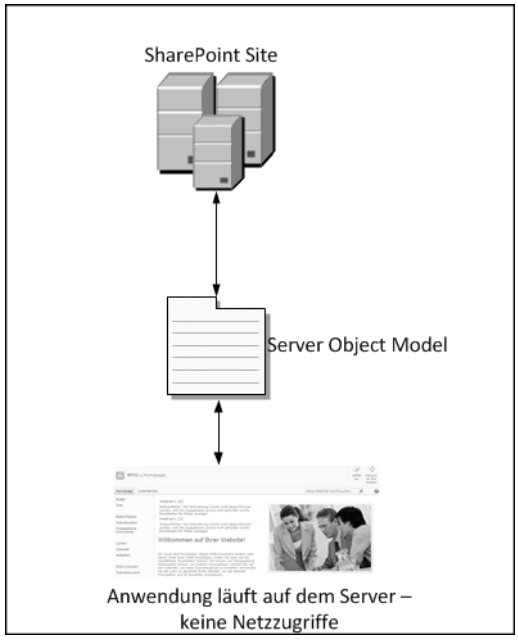


Das SharePoint Server-Objektmodell

Das Server Objektmodell entspricht dem alten SharePoint-Objektmodell. Das Modell bzw. seine Klassen können dazu verwendet werden, Anwendungen zu schreiben, die auf dem Server laufen, sei es als Befehlszeilenprogramme oder als Windows-Anwendungen. Die Entwicklung muss auf einem Server bzw. einem entsprechend konfigurierten Windows 7-Rechner erfolgen.

Die Architektur des SharePoint Server-Objektmodells sehen Sie in Abbildg. 26.4Abbildung 26.4.

Abbildg. 26.4 Architektur des SharePoint Server-Objektmodells



In Tabelle 26.2Tabelle 26.2 sind einige Klassen des Servermodells und des Clientmodells gegenüber gestellt. Die Systematik ist sofort erkennbar. Die Klassen unterscheiden sich in ihrer Benennung. Serverklassen haben das Präfix »SP«, Clientklassen nicht.

Tabelle 26.2 Gegenüberstellung Client- und Server-Objektmodell

Client	Server
Microsoft.SharePoint.Client.Site	Microsoft.SharePoint.SPSite
Microsoft.SharePoint.Client.Web	Microsoft.SharePoint.SPWeb
Microsoft.SharePoint.Client.List	Microsoft.SharePoint.SPList
Microsoft.SharePoint.Client.ListItem	Microsoft.SharePoint.SPListItem
Microsoft.SharePoint.Client.Field	Microsoft.SharePoint.SPField
Microsoft.SharePoint.Client.View	Microsoft.SharePoint.SPView
Microsoft.SharePoint.Client.User	Microsoft.SharePoint.Client.SPUser
Microsoft.SharePoint.Client.RoleAssignment	Microsoft.SharePoint.Client.SPRoleAssignment
Microsoft.SharePoint.Client.RoleDefintion	Microsoft.SharePoint.Client.SPRoleDefintion

Sie werden im Verlauf dieses Kapitels erkennen, dass sich die Programmierung auf Basis der beiden Modelle erheblich unterscheidet.

Erste Anwendung – Listen einer Website anzeigen

Das erste Einstiegsbeispiel soll so einfach wie möglich gehalten werden. Deshalb wurde eine Befehlszeilenanwendung gewählt, die alle Listen einer Websitesammlung anzeigen soll. Das Projekt wird Schritt für Schritt nachgestellt. Unser Anwendungsfall lässt sich grundlegend so beschreiben: »Zeige mir alle Listen der Website höchster Ebene in einer Websitesammlung.«

ACHTUNG Auch wenn die Aufgabenstellung relativ einfach erscheint, müssen doch mehrere Dinge berücksichtigt werden. Zum einen müssen Sie wissen, welche Objekte des Objektmodells Sie verwenden können, zum anderen, dass der Zugriff auf Listen nur über die Website möglich ist.

Wir brauchen also die Websitesammlung (die Website höchster Ebene) und den Zugriff auf deren Listen. Damit lassen sich die Objekte identifizieren.

Diese sind:

- SPSite – entspricht einer Websitesammlung
- SPWeb – entspricht einer Website
- SPList – entspricht einer Liste in einer Website

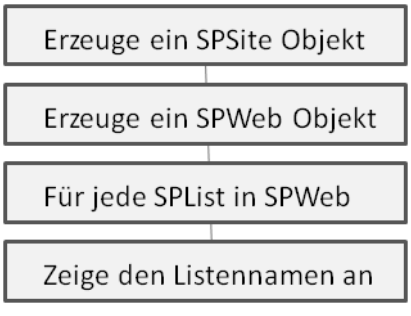
Als Entwickler setzen Sie daher die folgenden Schritte um:

1. Ermitteln der URL der auszulesenden Websitesammlung.
2. Anlegen des Projekts in Visual Studio.
3. Schreiben des Quellcodes in einer Anwendungsklasse.
4. Testen der Anwendung.

Um den Quellcode schreiben zu können, müssen Sie die Logik der Anwendung definieren.

Diese wurde in Abbildg. 26.5Abbildung 26.5 als aufeinander folgende Aktionen gezeichnet.

Abbildg. 26.5 Logik der ersten Anwendung – Listen einer Website anzeigen

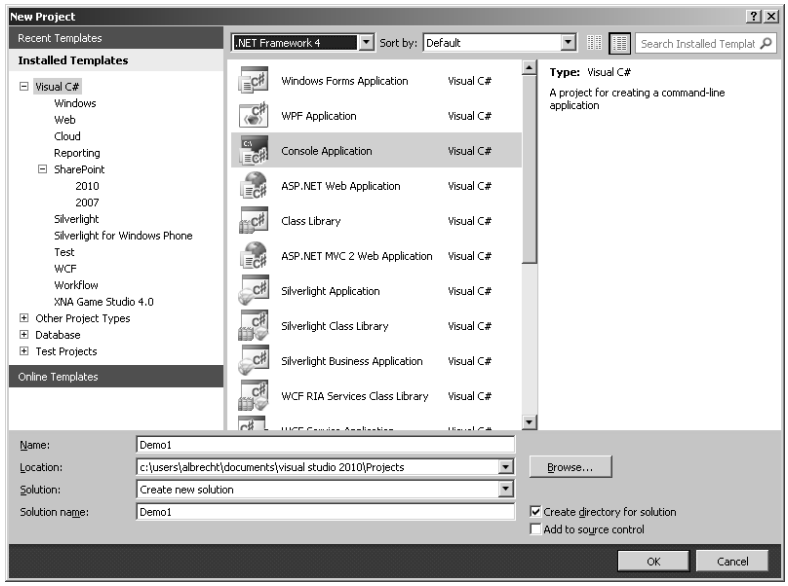


Starten Sie nun Visual Studio und legen Sie das Projekt folgendermaßen an:

1. Öffnen Sie Visual Studio und wählen Sie den Menübefehl *File/New/Project*.
2. Wählen Sie im Dialogfeld *New Project* die Vorlage **Console Application** unter Verwendung von Visual C# aus.
3. Geben Sie als Namen **Demo1** ein.

Sie sollten nun das in Abbildg. 26.6Abbildung 26.6 dargestellte Bild sehen.

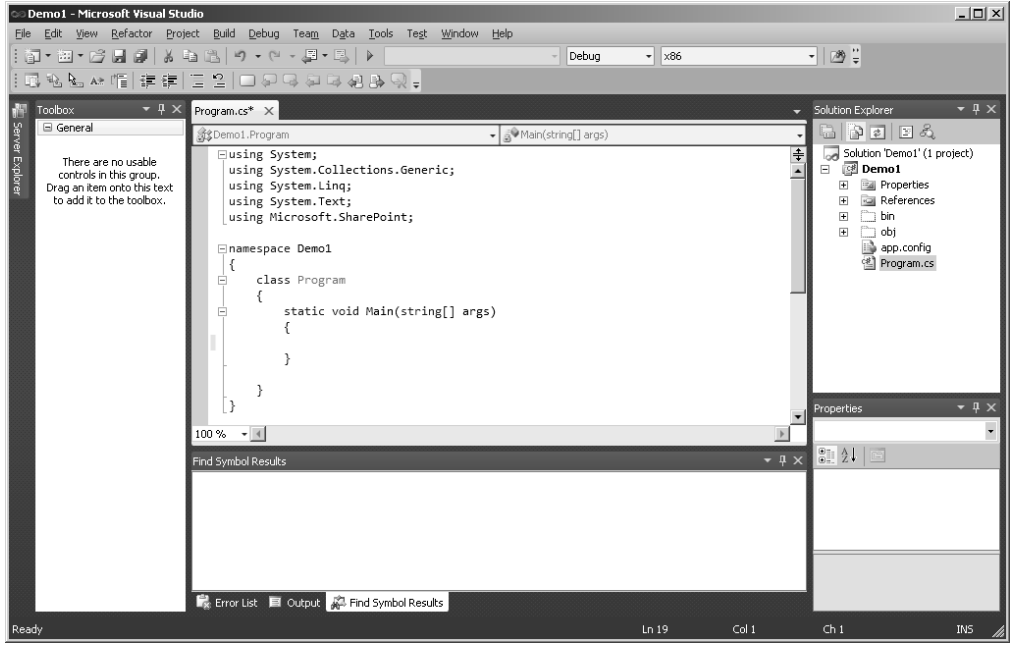
Abbildg. 26.6 Visual Studio – Projekt erstellen und Namen festlegen



4. Bestätigen Sie durch Klicken auf OK.

Das Projekt wird angelegt und Sie sehen in Visual Studio die in Abbildg. 26.7Abbildung 26.7 dargestellte Datei *Program.cs*. *Program.cs* ist in diesem Beispiel die ausführbare Klasse.

Abbildg. 26.7 Das neue Projekt im Visual Studio



HINWEIS Eine ausführbare Klasse erkennen Sie an der Methode »static void Main(string[] args)«.

Bevor wir mit der Beschreibung der eigentlichen Programmentwicklung beginnen, soll kurz die in Abbildg. 26.7Abbildung 26.7 dargestellte Entwicklungsumgebung erläutert werden.

- Toolbox

  Im linken Bereich sehen Sie das Fenster *Toolbox*. Bei grafischen Anwendungen werden hier die zur Verfügung stehenden Controls angezeigt, die dann per Drag & Drop in den Programmeditor gezogen werden können.

- Programmeditor

  Im mittleren Bereich sehen Sie oben den Programmeditor mit geöffneter Quellcodedatei. Auch hier ändert sich das Aussehen, wenn eine grafische Anwendung wie z.B. ein visuelles Webpart entwickelt wird. In diesem Fall kann der Entwickler in die Quellcode- oder Entwurfsansicht wechseln. Es besteht auch die Möglichkeit, das Fenster zu teilen, sodass in einem Fenster der Quellcode und im anderen Fenster der grafische Entwurf angezeigt werden kann.

- Ausgabefenster

  Im unteren Bereich wird das Ausgabefenster (Output) angezeigt, indem z.B. die Statusausgaben beim Übersetzen einer Anwendung gezeigt werden.

- Solution Explorer

  Im rechten Bereich sehen Sie den Solution oder Projekt-Explorer, der alle am Projekt beteiligten Dateien in einer Baumstruktur anzeigt.

- Properties

  Die rechte untere Hälfte wird vom Properties- oder Eigenschaftsfenster ausgefüllt.

HINWEIS Visual Studio ist eine sehr flexible Entwicklungsumgebung und kann individuell auf die eigenen Bedürfnisse angepasst werden. Die unterschiedlichen Arbeitsbereiche oder Fenster finden Sie als Optionen im Menü *View*.

Zu Beginn eines Programmierprojekts müssen die Eigenschaften der Anwendung konfiguriert werden. Bei der Entwicklung von SharePoint 2010-Anwendungen sind das die Einstellungen zum .NET Framework und die Einstellung, für welches Betriebssystem bzw. welche Prozessorarchitektur das Programm übersetzt werden soll. Für SharePoint 2010 ist das immer eine 64-Bit-Architektur mit .NET Framework Version 3.5. Visual Studio 2010 konfiguriert eine Anwendung aber standardmäßig nicht mit diesen Einstellungen.

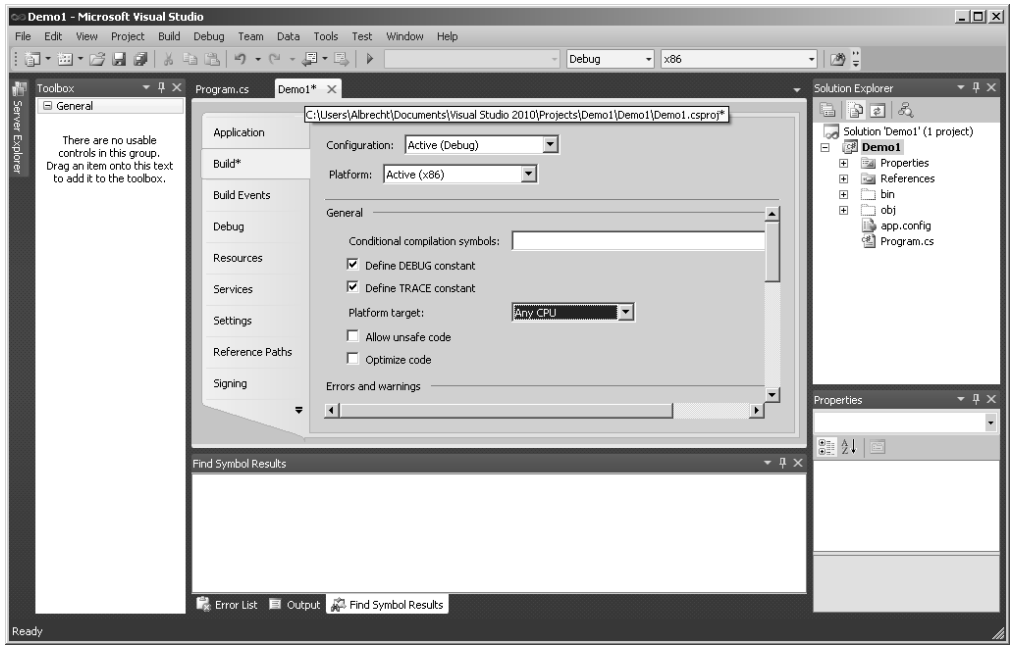
HINWEIS Wenn Sie versuchen, eine SharePoint-Anwendung mit anderen Einstellungen zu erzeugen, dann zeigt die Entwicklungsumgebung nicht nachvollziehbare Fehlermeldungen.

So gehen Sie vor, um das Programmierprojekt zu starten:

1. Öffnen Sie das Kontextmenü des Projektnamen und wählen Sie *Properties*. Alternativ können Sie auch, nach Markieren des Projektnamens, die Tastenkombination Alt + ↵ verwenden.

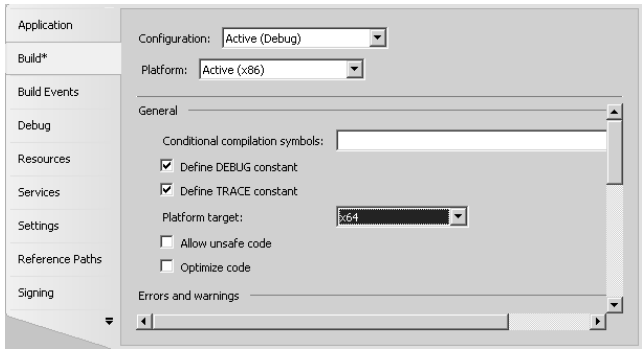
Sie sehen das in Abbildg. 26.8Abbildung 26.8 gezeigte Dialogfeld, mit dessen Hilfe die Konfiguration einer Anwendung beschrieben wird.

Abbildg. 26.8 Eigenschaften (Properties) einer SharePoint-Anwendung



2. Öffnen Sie die Registerkarte Application und wählen Sie im Dropdownmenü *Target Framework* den Eintrag *.NET Framework 3.5*. Sie werden darauf Aufmerksam gemacht, dass das Projekt neu geladen wird.
3. Öffnen Sie nach dem Laden erneut die Eigenschaften des Projekts. Wechseln Sie in die Registerkarte *Build* und legen Sie im Dropdownmenü *Target Platform* den Eintrag »x64« fest (Abbildg. 26.9Abbildung 26.9).

Abbildg. 26.9 Zielplattform ändern



4. Schließen Sie das Dialogfeld mit den Eigenschaften.

Wenn Sie den Quellcode in Listing 26.1Tabelle 26.1 betrachten, dann lässt nichts darauf schließen, dass es sich um eine SharePoint-Anwendung handelt. Dennoch sehen Sie ein lauffähiges Programm vor sich. Wenn Sie es starten wird genau das passieren, was programmiert wurde – nämlich nichts.

Listing 26.1 Eine *leere* Anwendung

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Demo1
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Als erstes müssen Sie jetzt das Server-Objektmodell einbinden, das heißt die Klassen, die den Zugriff auf SharePoint zur Verfügung stellen. Dazu ist es notwendig, die Assembly zu referenzieren. Mit Hilfe der Eigenschaften und Methoden der Klassen des Server-Objektmodells kann man dann auf die Elemente von SharePoint *direkt* zugreifen.

HINWEIS *Direkt* bedeutet in diesem Fall, dass der Zugriff nicht über ein Netzwerk erfolgt.

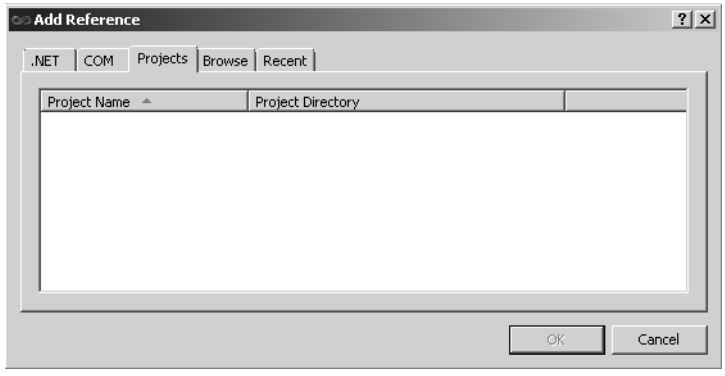
So gehen Sie vor, um die Assembly zu referenzieren:

1. Markieren Sie in Visual Studio im Solution Explorer den Knoten *References*.

ACHTUNG Der Platzhalter *%Programmverzeichnis%* steht als Variable für das Programmverzeichnis des Servers, z.B. *C:\Programme*. Je nach Konfiguration kann das Programmverzeichnis auf Ihrem System in einem anderen Pfad liegen.

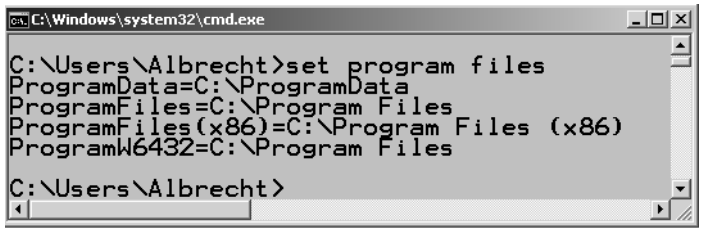
2. Öffnen Sie das Kontextmenü und wählen Sie *Add References*. Sie sehen in Abbildg. 26.10Abbildung 26.10 das sich öffnende Dialogfeld.
3. Wechseln Sie in das Verzeichnis, in dem bei der Installation von SharePoint das Server-Objektmodell angelegt wurde. Das ist in der Regel das Verzeichnis:
   %Programmverzeichnis%\Common Files\Microsoft Shared\Web Server Extensions\14\ISAPI.

Abbildg. 26.10 Referenz auf SharePoint-Objektmodell einbinden



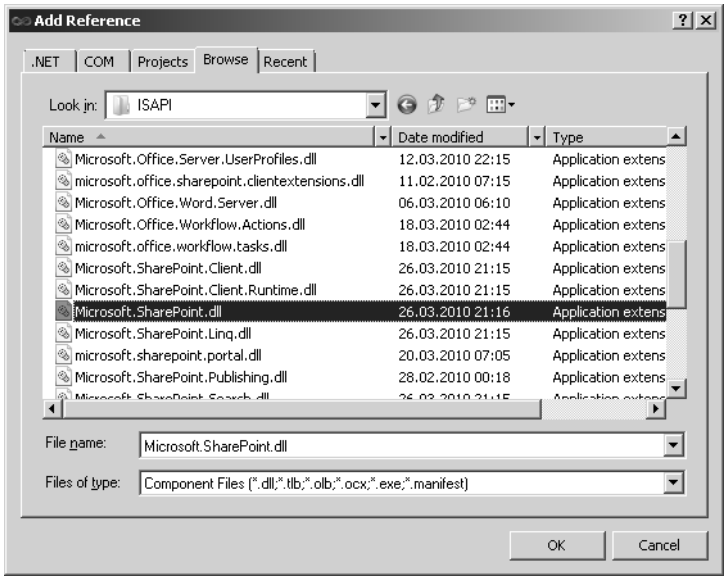
TIPP Sie können das Programmverzeichnis schnell ermitteln, indem Sie ein Befehlszeilenfenster öffnen, den Befehl **set ProgramFiles** eingeben und mit ↵ bestätigen. Abbildg. 26.11Abbildung 26.11 zeigt das Ergebnis des Befehls auf dem Autoren-Entwicklungsrechner.

Abbildg. 26.11 Anzeigen des Programmverzeichnisses



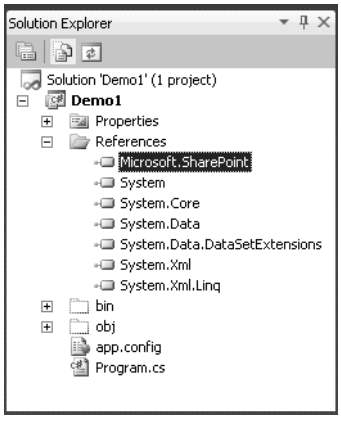
4. Nachdem Sie das Programmverzeichnis ermittelt haben, klicken Sie auf *Browse* und navigieren Sie in das entsprechende Verzeichnis.
5. Markieren Sie, wie in der Abbildg. 26.12Abbildung 26.12 dargestellt, im Listenfeld des Dialogfelds Add Reference den Eintrag Microsoft.SharePoint.dll und klicken Sie auf OK.

Abbildg. 26.12 SharePoint Assembly auswählen



Die Referenz auf die Assembly ist jetzt im Solution Explorer im Knoten *References* zu sehen. (vgl. Abbildung Abbildg. 26.1326.13)

Abbildg. 26.13 Eingebundene Referenz auf das Server-Objektmodell



Als Ergebnis können Sie nun auf die Klassen zugreifen, die für den Zugriff auf den SharePoint benötigt werden.

6. Ergänzen Sie den Quellcode der Datei *Program.cs* aus Listing 26.1, wie in Listing 26.2 gezeigt.

Listing 26.2 Programm mit Verweis auf SharePoint-Objekte

```
using System;
using System.Collections.Generic;
```

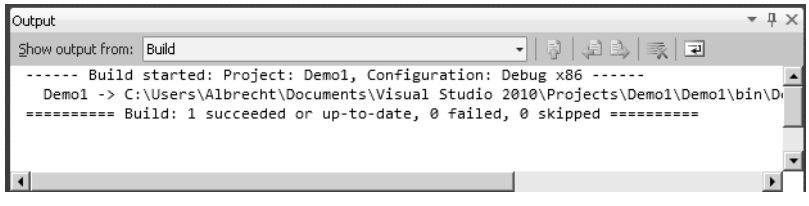
Listing 26.2 Programm mit Verweis auf SharePoint-Objekte

```
using System.Linq;
using System.Text;
using Microsoft.SharePoint;

namespace Demo1
{
    class Program
    {
        static void Main(string[] args)
        {
            string url = "http://localhost";
            SPSite sitecollection = new SPSite(url);
            SPWeb web = sitecollection.RootWeb;

            foreach (SPList liste in web.Lists)
            {
                Console.WriteLine(liste.Title);
            }

            Console.ReadLine();
        }
    }

}
```

7. Überprüfen Sie, ob Ihr Quellcode fehlerfrei ist, indem Sie das Kontextmenü des Projektnamen öffnen und den Eintrag *Build* wählen. Damit wird das Programm kompiliert.

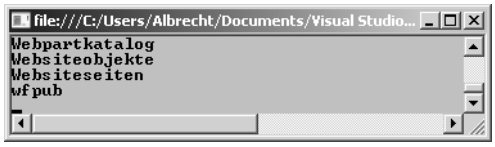
Im Visual Studio-Ausgabefenster Output sehen Sie bei einem fehlerfreien Quellcode das in Abbildg. 26.14Abbildung 26.14 dargestellte Ergebnis. Andernfalls werden die Fehler mit einem entsprechenden Verweis auf die Programmzeile aufgelistet. Mit einem Doppelklick können Sie direkt in die Zeile wechseln und den Fehler korrigieren.

Abbildg. 26.14 Output nach dem Kompilieren



8. Wurde der Quellcode fehlerfrei übersetzt, klicken Sie auf F5. Die Anwendung wird von Visual Studio gestartet. In einem Konsolenfenster können Sie jetzt alle Listen der Websitesammlung sehen.

Abbildg. 26.15 Ausgabe des Programms *Demo 1*



Wenn Sie nun die Anwendung analysieren, dann sehen Sie die Verwendung von drei SharePoint-Klassen, nämlich:

- SPSite
- SPWeb
- SPList.

Hier müssen Sie folgendes beachten:

- Eine Websitesammlung (SiteCollection) ist im Programmiermodell eine SPSite.
- Eine Website ist im Programmiermodell ein SPWeb.

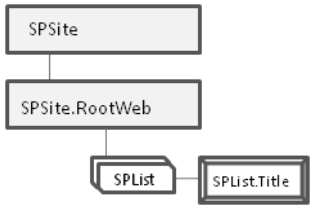
In einer Websitesammlung gibt es immer ein Web als Website höchster Ebene, die im Modell »RootWeb« genannt wird und eine Eigenschaft der SPSite ist.

Eine SPWeb wiederum kann beliebig viele weitere SPWebs enthalten.

WICHTIG Das Verständnis dieser Zusammenhänge ist für die SharePoint-Programmierung von fundamentaler Bedeutung.

Abbildg. 26.16Abbildung 26.16 illustriert den hierarchischen Aufbau der Klassen.

Abbildg. 26.16 Hierarchischer Aufbau der SharePoint Server-Klassen



In unserem Beispiel werden alle Listen der Site höchster Ebene angezeigt. Dazu wird mit der folgenden Zeile

```
SPSite sitecollection = new SPSite(url);
```

ein SPSite Objekt erzeugt.

Als Parameter wird die URL der Websitesammlung, im gezeigten Beispiel ist das die Adresse *http://localhost*, übergeben Die Websitesammlung liegt also im Stammverzeichnis der Webanwendung.

Mit der nächsten Zeile wird ein SPWeb-Objekt erzeugt.

```
SPWeb web = sitecollection.RootWeb;
```

Das Objekt entspricht der Website höchster Ebene der Websitesammlung. In der nachfolgenden Schleife werden nach und nach alle Listen (SPList) des Webs durchlaufen.

```
foreach (SPList liste in web.Lists)
{
                Console.WriteLine(liste.Title);
}
```

In der Schleife selbst wird die jeweilige »Title-Eigenschaft« der Liste auf der Konsole ausgegeben. Die Title-Eigenschaft der Liste bildet den Namen der Liste ab.

> Der Kopf der Schleife kann wie folgt gelesen werden:
>
> »Erzeuge für jede Liste in der Sammlung aller Listen der Website `web` ein Listenobjekt. Wiederhole dies solange, wie es noch Listen in der Listensammlung `web.Lists` gibt.«

Diese Schleifenkonstruktion wird Ihnen in den folgenden weiteren Programmbeispielen immer wieder begegnen.

Was ist aber nun, wenn Sie nicht nur die Listen einer Website anzeigen möchten sondern die Listen aller Websites der Websitesammlung?

Listing 26.3 zeigt deshalb eine Erweiterung des vorliegenden Programms, die nochmals den hierarchischen Aufbau des SharePoint Server und damit auch des Objektmodells verdeutlicht.

Hier werden von allen Websites der Websitesammlung die Listen ausgeben. Die Ergänzungen sind im Quellcode entsprechend kommentiert. Das Geheimnis liegt hier in der Verschachtelung zweier Schleifen. In der äußeren Schleife werden alle Websites abgearbeitet, in der inneren die Listen der jeweiligen Website.

Listing 26.3 Alle Listen aller Websites ausgeben

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.SharePoint;

namespace Demo1
{
    class Program
    {
        static void Main(string[] args)
        {
            string url = "http://localhost";
            SPSite sitecollection = new SPSite(url);
            //  wird nicht mehr benötigt, weil ich alle Webs durchlaufe
            //   SPWeb web = sitecollection.RootWeb;
```

Listing 26.3 Alle Listen aller Websites ausgeben

```
            //neu
            SPWebCollection alleWebs = sitecollection.AllWebs;


// neue Schleife, ich durchlaufe alle Website und dann in einer Website die Listen
            foreach (SPWeb dasWeb in alleWebs)
            {

                // diese Zeilen dienen zur besseren Formatierung der Konsoleausgabe
                Console.WriteLine(dasWeb.Title);
                Console.WriteLine("------------------------");
                // ende

                // Schleife in der Schleife, wird für jedes Website durchlaufen
                foreach (SPList liste in dasWeb.Lists)
                {
                    Console.WriteLine(liste.Title);
                }

                // Leerzeile nach der Auflistung der Listen eines Web
                Console.WriteLine();

            }
            Console.ReadLine();

        }

    }
}
```

Im Quellcode finden Sie eine Klasse, die Sie noch nicht kennen – die »SPWebCollection«. Die Klasse repräsentiert die Sammlung aller Websites einer Websitesammlung. Mit Hilfe der Eigenschaft »AllWebs von SPSite« kann die SPWebCollection gefüllt werden.

Die Sammlung (Collection) enthält dann alle Websites – unabhängig davon, wo diese in der Sitehierarchie zu finden sind.

HINWEIS Legen Sie mindestens eine weitere Website in der Website höchster Ebene an. Wenn Sie dies nicht tun, dann entspricht die Ausgabe der Anwendung der Ausgabe der ersten Version. In Abbildg. 26.17Abbildung 26.17 ist zu erkennen, dass die Websitesammlung zwei Websites beinhaltet – das »RootWeb« und eine Website mit dem Namen »KalenderSite«.

Abbildg. 26.17 Ausgabe der zweiten Variante von *Demo1*

Zweite Anwendung – Listenelemente anzeigen

Das zweite Programmbeispiel in diesem Kapitel ist etwas komplexer. Es sollen jetzt ausgewählte Metadaten in einer Liste anzeigt werden. Dazu wird ein Kalender ausgelesen, der ja nichts anderes ist als eine Liste, um alle Termine mit Anfangs- und Enddatum anzeigen zu lassen.

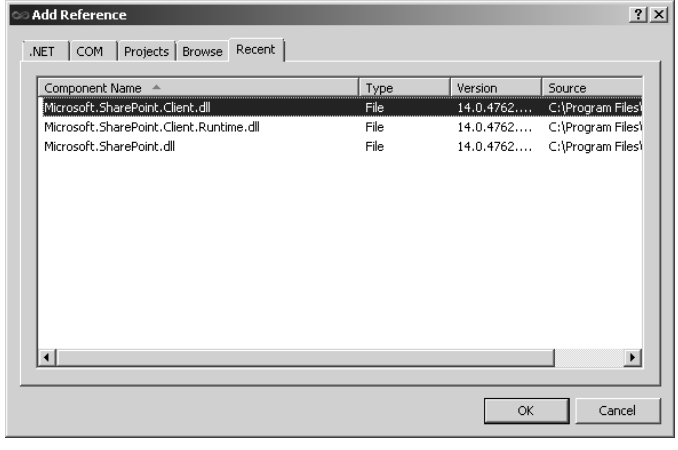
HINWEIS Die Begriffe »Metadaten«, »Spalte« und später auch »Feld« werden synonym verwendet. Diese Begriffe spiegeln einfach nur eine jeweils andere Sichtweise wider. Aus der Sicht des Anwenders spricht man von Spalte. Für Administratoren sind Spalten Metadaten und für Entwickler sind es Felder.

Die Programmlogik des Projekts ist die gleiche wie im ersten Beispiel. Allerdings werden hier, neben der Adresse der Websitesammlung, auch den Namen des Kalenders übergeben.

TIPP Mit einem kleinen Trick können Sie das Einbinden der Referenz beschleunigen.

- Öffnen Sie dazu das Dialogfeld *References Add* und anschließend das Register *Recent*.
- Wählen Sie die Assembly, siehe Abbildg. 26.18Abbildung 26.18, und bestätigen Sie mit OK.

Abbildg. 26.18 Registerkarte Recent des Add References Dialogfelds



Für das Anwendungsbeispiel wurde eine weitere Website unterhalb des RootWebs, also der Site höchster Ebene, angelegt. Das ist in unserem Modell ein SPWeb im RootWeb der SPSite. Deshalb wird der Name des Webs zu einem späteren Zeitpunkt noch benötigt.

HINWEIS Gerade als Einsteiger kann es sehr verwirrend sein, das mit SPWeb immer eine Website und mit SPSite immer eine Websitesammlung gemeint ist. Lassen Sie sich nicht beirren. Die Erläuterungen finden Sie auf Seite 841.

Die Vorbereitungen sind abgeschlossen. So gehen Sie vor, um das Programmierbeispiel nachzustellen:

1. Legen Sie ein neues Projekt an. Geben Sie dem Projekt den Namen »Demo2«. (s. Seite 842) Vergessen Sie auch nicht, die Projekteigenschaften anzupassen, die standardmäßig nicht zu einem SharePoint-Projekt passen.

Das Besondere an diesem Beispiel ist, dass auf Metadaten einer Liste zugegriffen wird. Dazu muss die Bezeichnung dieser Metadaten (Spalten) bekannt sein.

HINWEIS Wie Sie die Bezeichnungen der Metadaten ermitteln können, haben Sie im Kapitel 3 »Dokumentbibliotheken« und 4 »Listen« kennen gelernt. Sie finden die Spaltenbezeichnungen in den jeweiligen Listeneinstellungen.

Die Kalenderspalten haben folgende Anzeigenamen:

- Titel
- Anfangszeit
- Endzeit.

Der interne Name ist jeweils »Title«, »EventDate« und »EndDate«.

HINWEIS Warum sowohl der Anzeigename als auch der interne Name von Bedeutung sein können, werden Sie im Zusammenhang mit dem Client Objektmodell erfahren. In manchen Zusammenhängen wird der interne Name auch als statischer Name bezeichnet. Das verweist darauf, dass dieser Name nicht geändert werden kann.

TIPP Wir raten Ihnen dringend, beim Anlegen von Metadaten zunächst einen Namen zu verwenden, der nur aus lateinischen Schriftzeichen besteht, z.B. »geloescht«. Der Hintergrund ist folgender:

- Der zuerst verwendete Name wird von SharePoint als interner Name angelegt und nicht mehr verändert.
- Nachdem die Spalte angelegt wurde, können Sie den Namen jederzeit ändern, z.B. in »gelöscht«.
- Damit ändern Sie den Anzeigenamen der Spalte, der interne Name bleibt unverändert.

Wenn Sie lateinische Schriftzeichen verwenden, bleibt der interne Name für Sie lesbar. Wenn Sie Umlaute verwenden, werden diese von SharePoint intern in Sonderzeichen umgewandelt. Im Falle der Verwendung des internen Namens »gelöscht« würde SharePoint das Feld im Code umbenennen in »gel%5Fx00f6%5Fscht«. Der Verweis in einer Anwendung wird in diesem Fall wesentlich schwieriger.

Im Folgenden sehen Sie das Listing des Beispielprogramms.

2. Geben Sie den in Listing 26.4 gezeigten Quellcode ein.

Listing 26.4 Programmcode mit Zugriff auf Metadaten

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.SharePoint;

namespace Demo2
{
    class Program
    {
        static void Main(string[] args)
        {
            string url = "http://localhost";
            string subWeb = "KalenderSite";
            string kalenderName = "Meinetermine";

            SPSite sitecollection = new SPSite(url);
            SPWeb web = sitecollection.RootWeb;
            SPWeb kalenderWeb = web.Webs[subWeb];
            SPList kalender = kalenderWeb.Lists[kalenderName];

            foreach (SPItem item in kalender.Items)
            {

                Console.WriteLine(item["Titel"] + "\t\t" + item["Anfangszeit"] + "\t\t"
                + item["Endzeit"]);
            }

            Console.ReadLine();
        }

        }
    }
```

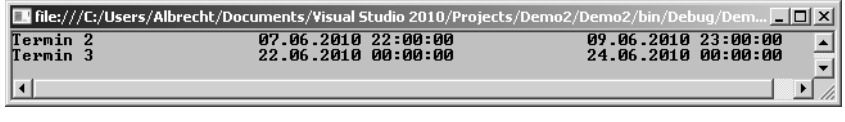
Im Listing 26.4Listing 26.4 lernen Sie eine weitere Klasse des Server Objektmodells kennen, die SPItem Klasse.

»SPItem« entspricht den Zeilen einer Liste.

Mit diesem Wissen kann die Schleife analysiert werden. Hier wird jeweils ein SPItem-Objekt erzeugt, indem die Liste aller Items der Kalenderliste durchlaufen wird. In der Schleife selbst werden die Metadaten oder Spalten mit den Namen »Titel«, »Anfangszeit« und »Endzeit« ausgegeben. Zur besseren Darstellung werden die Spalten durch zwei Tabulatoren getrennt.

HINWEIS Das Programmbeispiel zeigt, dass der Zugriff auf die Metadaten mit dem Anzeigename erfolgen kann. Es gibt aber Situationen, in denen Sie auf jeden Fall auf den internen Namen zugreifen müssen.

Abbildg. 26.19 Ausgabe des Projekts *Demo2*



Mit den beiden Programmen *Demo1* und *Demo2* haben Sie an einfachen Beispielen gesehen, wie die Hierarchie von SharePoint Server programmtechnisch abgefragt und analysiert werden kann. Von der Websitesammlung kommt man zu den Websites, hier zu den Listen und von den Listen zu den einzelnen Elementen oder Zeilen der Listen.

Eine Hierarchieebene fehlt noch, nämlich die der Metadaten oder Spalten. Diese Ebene werden Sie nun im nächsten Programm kennen lernen, welches dann die Einführung in die grundlegenden Objekte des Server Objektmodells abschließt.

Dritte Anwendung – alle Spalten einer Liste anzeigen

So gehen Sie vor, um alle Spalten einer Liste anzuzeigen:

1. Legen Sie ein weiteres neues Projekt an und geben Sie ihm den Namen *Demo3*. (s. Seite 842)

HINWEIS Das Neue an diesem Beispiel ist die Umsetzung eines Problems, das durchaus häufiger auftritt. In diesem Szenario ist nur der Name einer Liste, nicht jedoch die der Spalten bekannt. Der Anwendungsfall lässt sich folgendermaßen beschreiben:

»Zeige den Anzeigenamen und den internen Namen aller sichtbaren Spalten der Kalenderliste an.«

Zur Umsetzung des Programms müssen Sie also wissen, wie im Objektmodell das Objekt heißt, dass die Spalten eines SPItems repräsentiert.

Das Objekt heißt »SPField«. Auch hier können Sie erkennen, dass das Objektmodell zwar den SharePoint abbildet, aber andere Bezeichnungen wählt. Im SharePoint werden die SPFields nicht Felder sondern »Spalten« oder »Metadaten« genannt.

Die Zeilen aus Listing 26.5 zeigen das Programm. Vieles wird Ihnen nun schon vertraut erscheinen. Das Programm entspricht im Aufbau den bereits beschriebenen Programmen.

2. Geben Sie den in Listing 26.5Listing 26.5 gezeigten Quellcode ein.

Listing 26.5 Programmcode zur Ausgabe von Anzeigenamen und internen Namen

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.SharePoint;


namespace Demo3
{
    class Program
    {
        static void Main(string[] args)
        {
            string url = "http://localhost";
            string subWeb = "KalenderSite";
            string kalenderName = "Meinetermine";

            // diese Variablen dienen der Druckaufbereitung
            string internerName = "";
            string anzeigeName = "";
```

Listing 26.5 Programmcode zur Ausgabe von Anzeigenamen und internen Namen

```
            // ende

            SPSite sitecollection = new SPSite(url);
            SPWeb web = sitecollection.RootWeb;
            SPWeb kalenderWeb = web.Webs[subWeb];
            SPList kalender = kalenderWeb.Lists[kalenderName];

            foreach (SPField field in kalender.Fields)
            {
                if (field.Hidden == false)
                {
                    internerName = field.StaticName;
                    anzeigeName = field.Title;

                    Console.WriteLine(internerName.PadRight(25) + anzeigeName);
                }
            }
            Console.ReadLine();

        }
    }
}
```

Neu ist das »SPField Objekt« und die Fields-Eigenschaft der Liste. Ebenfalls neu ist die Verzweigung innerhalb der Schleife – die »if-Anweisung«.

```
if (field.Hidden == false)
{
                    internerName = field.StaticName;
                    anzeigeName = field.Title;
                    Console.WriteLine(internerName.PadRight(25) + anzeigeName);
}
```

Hier wird überprüft, ob das Feld (also die Spalte), versteckt ist oder nicht. Spalten mit der Eigenschaft »Hidden gleich true« können in den Ansichten von SharePoint nicht angezeigt werden. Auf Programmebene kann aber auf diese Spalten zugegriffen werden. Im vorliegenden Beispiel werden nur Felder angezeigt, die nicht versteckt sind.

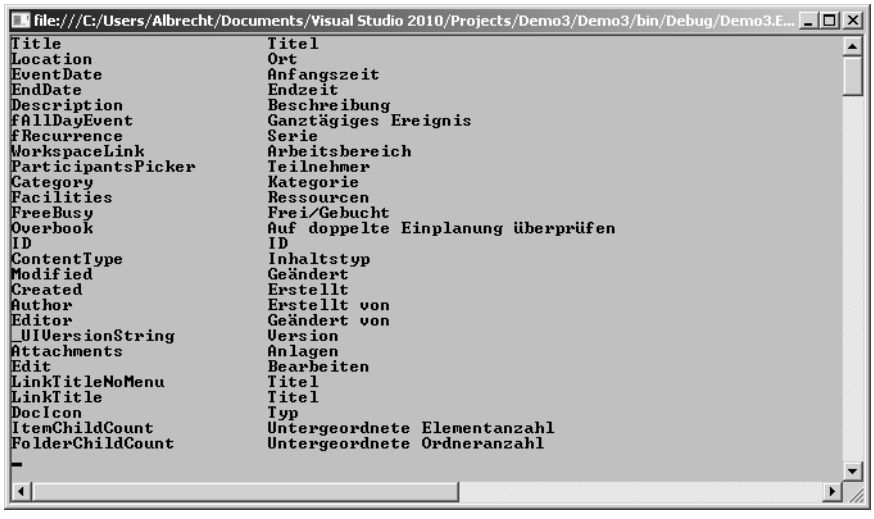
Objekte der Klasse »SPField« haben die Eigenschaften »Title« und »StaticName«, wobei »Title« für den Anzeigenname steht und »StaticName« für den internen Namen.

HINWEIS Hier wird wieder deutlich, dass das Objektmodell zwar den SharePoint abbildet, aber in vielen Bereichen mit anderen Namen arbeitet. Der Grund für die nicht einheitliche Verwendung der Namen von Objekten liegt in der Entwicklungsgeschichte von SharePoint. Was in der aktuellen Version eine Site oder Website ist, wurde in den ersten Versionen *Web* genannt.

Es ist leichter, die neuen Namen zu verwenden als ein Objektmodell umzuschreiben. Wenn Sie also in die SharePoint-Entwicklung einsteigen möchten, sollten Sie wissen, wie welche SharePoint-Elemente im Objektmodell bezeichnet werden.

Die Funktion »PadRight(25)« wird genutzt, um die beiden Feldeigenschaft in zwei Spalten auszurichten. Das Ergebnis zeigt Abbildg. 26.20Abbildung 26.20.

Abbildg. 26.20 Ausgabe des Projekts *Demo3*



Ändern Sie das Programm und entfernen Sie die Verzweigung. Als Ergebnis werden dann alle Felder aufgelistet. Sie werden erstaunt sein, wie viele Felder eine Kalenderliste tatsächlich besitzt.

Vierte Anwendung – ein neues Listenelement erstellen

Bis jetzt haben Sie gesehen, wie man SharePoint-Listen analysieren und auslesen kann. Nun zeigen wir Ihnen, wie Sie Daten in eine Liste schreiben können. Der hier umgesetzte Anwendungsfall lässt sich wie folgt beschreiben:

»Trage in eine Kalenderliste einen neuen Termin ein.«

Legen Sie ein weiteres neues Projekt an und geben Sie ihm den Namen *Demo4*. (s. Seite 842). Das Besondere an diesem Beispiel ist, dass Sie in einer Liste eine neue Zeile erzeugen und dann die Spalten der Zeile bearbeiten müssen.

HINWEIS Aus Anwendersicht besteht eine Liste aus Zeilen. Aus Entwicklersicht besteht eine Liste aus »Items«.

Aufgrund der Vorgabe für das Projekt *Demo4* müssen Sie jetzt nicht nur mit den Eigenschaften einer Liste arbeiten, sondern auch mit Funktionen, über die eine Liste verfügt. Die Funktionen werden auch Methoden genannt.

HINWEIS An dieser Stelle ist es sinnvoll, sich etwas näher mit einer SharePoint-Klasse zu beschäftigen. Als Beispiel bietet sich die »SPList« an. Markieren Sie in einem vorliegenden Quellcode SPList und drücken Sie dann F12. Als Ergebnis wird die Definition der Klasse geladen. Bei der Analyse wird deutlich, wie mächtig diese Klasse ist. Sie verfügt über 53 Methoden und sage und schreibe 129 Eigenschaften. Man könnte also allein über diese Klasse ein ganzes Buch schreiben.

Die für unser Beispiel benötigte Funktion heißt »AddItem()«, die zu einer Liste ein neues Item hinzufügt.

Diese Methode gehört damit zum Verhalten einer Liste. Methoden erkennen Sie immer an den geschweiften Klammern. Eigenschaften werden immer ohne Klammern notiert.

HINWEIS Bei den Eigenschaften handelt es sich im engeren Sinne um »Properties«. Properties zeichnen sich dadurch aus, dass Sie gelesen und/oder geschrieben werden können.

Im Quellcode sieht das dann wie in den folgenden Beispielen aus der Klasse »SPList« aus.

Nachfolgend sehen Sie die Eigenschaften, die man nur lesen kann, wie die Anzahl der Items, und eine Eigenschaft, die man auch verändern kann, wie z.B. den Titel, das heißt den Namen der Liste.

```
public int ItemCount { get; }
public string Title { get; set; }
```

Die Zeilen in Listing 26.6 zeigen das fertige Programm. Die für Sie neuen Zeilen sind mit dem Kommentar »neu« und »ende neu« eingebunden. Geben Sie den in Listing 26.6Listing 26.6 gezeigten Quellcode ein.

Listing 26.6 Metadaten schreiben

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.SharePoint;

namespace Demo4
{
    class Program
    {

        static void Main(string[] args)
        {

            string url = "http://localhost";
            string subWeb = "KalenderSite";
            string kalenderName = "Meinetermine";

            SPSite sitecollection = new SPSite(url);
            SPWeb web = sitecollection.RootWeb;
            SPWeb kalenderWeb = web.Webs[subWeb];
            SPList kalender = kalenderWeb.Lists[kalenderName];


            // neu
            SPItem neuesElement = kalender.AddItem();

            neuesElement["Titel"] = "Termin 3";
            neuesElement["Anfangszeit"] = new DateTime(2010, 6, 22);
            neuesElement["Endzeit"] = new DateTime(2010, 6, 24);

            neuesElement.Update();
```

Listing 26.6 Metadaten schreiben *(Fortsetzung)*

```
            // ende neu
            Console.WriteLine("Neuer Termin wurde eingetragen");
        }


    }
}
```

Da die Logik des Programmvorgangs linear ist, ist der Quellcode einfach zu verstehen:

- In der ersten Zeile wird ein »SPItem«-Objekt, welches das Ergebnis der Funktion oder Methode »AddItem()« ist, erzeugt.
- AddItem() erzeugt also nicht nur ein neues Items, sondern stellt diese als Ergebnis zur weiteren Verarbeitung zur Verfügung.

Wenn also jetzt im weiteren Verlauf den Spalten von »neuesElement« Werte zugewiesen werden, wird das neue Item der Kalenderliste bearbeitet.

- Um die Änderungen zu speichern, schließen sie den Vorgang mit der Methode »Update()« ab.

HINWEIS Beachten Sie, das »Update()« eine Methode des Items und »AddItem()« eine Methode der Liste ist. Wenn Sie die Methode »Update()« nicht verwenden, würde zwar ein neues Item erzeugt werden, die Änderungen in den Spalten aber würden nicht gespeichert werden.

- Starten Sie die Anwendung. Es wird eine kurze Meldung ausgegeben, dass ein neuer Termin eingefügt wurde.

Dem Beispielprogramm fehlt natürlich jede Interaktivität. Es ist nicht möglich, beliebige Termine einzugeben. Dazu fehlen Komponenten, die die Eingaben des Benutzers entgegennehmen und verarbeiten.

Fünfte Anwendung – ein Listenelement suchen und anzeigen

Bis jetzt haben Sie die Programmierung der grundlegenden Anwendungsoperationen wie Anzeigen, Erstellen und Ändern kennen gelernt. Was jetzt noch fehlt ist das Suchen und auch das Löschen von Items. Zunächst zeigen wir Ihnen, innerhalb des Projekts 5, wie Sie den folgenden Anwendungsfall umsetzen können.

»Suche Anfangs- und Endezeit des Termins Termin 3.«

Um diesen Anwendungsfall umzusetzen, müssen Sie die Kalenderliste durchsuchen und das Ergebnis anzeigen. Das Beispielprogramm muss also zwei Prozesse umsetzten – einmal das Suchen als solches und zum anderen die Verarbeitung des Suchergebnisses.

Von diesem Ergebnis wissen Sie nicht, ob es ein, mehrere oder gar keinen Treffer gibt.

HINWEIS Für Einsteiger oder interessierte SharePoint-Anwender ist dieses Programm relativ komplex, weil die Suche auf einer XML-Syntax basiert. Wichtig ist deshalb, das Programm grundlegend zu verstehen.

1. Legen Sie nun das Projekt *Demo5* an. (siehe Seite 842).
2. Geben Sie den in Listing 26.7Listing 26.7 gezeigten Quellcode ein.

Listing 26.7 Listenelemente suchen und anzeigen

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.SharePoint;

namespace Demo5
{
    class Program
    {
        static void Main(string[] args)
        {
            string url = "http://localhost";
            string subWeb = "KalenderSite";
            string kalenderName = "Meinetermine";
            SPSite sitecollection = new SPSite(url);
            SPWeb web = sitecollection.RootWeb;
            SPWeb kalenderWeb = web.Webs[subWeb];
            SPList kalender = kalenderWeb.Lists[kalenderName];

            // Abfrage erzeugen und starten
            SPQuery abfrage = new SPQuery();
            string caml = "<Where><Eq><FieldRef Name='Title'/>" +
                          "<Value Type='Text'>Termin 3</Value></Eq></Where>";
            //string caml = "<Where><Eq><FieldRef Name='EventDate'/><Value
            //Type='DateTime'>2010-06-22T10:00:00Z</Value></Eq></Where>";

            abfrage.Query = caml;

            // Ergebnis speichern und Auswerten
            SPListItemCollection ergebnisse = kalender.GetItems(abfrage);
            // wenn wir nichts gefunden haben

            if (ergebnisse.Count < 1)
            {
                Console.WriteLine("Termin mit der Bezeichnung Termin 3 nicht gefunden");
                return;
            }
            // alle gefundenen SPitems anzeigen
            foreach (SPItem item in ergebnisse)
            {

                Console.WriteLine(item["Titel"] + "\t\t" + item["Anfangszeit"]
                                  + "\t\t" + item["Endzeit"]);
            }
            Console.ReadLine();
        }
    }
}
```

Drei Dinge sind für das Gesamtverständnis von entscheidender Bedeutung:

- die Klasse SPQuery,
- der CAML Ausdruck im String »caml« und
- die Methode »GetItems()« der SPList-Klasse.

HINWEIS Wie der Name es schon andeutet, wird über SPQuery eine Abfrage erzeugt. Diese Abfrage benutzt einen XML-Dialekt, der CAML heißt. CAML steht für **C**ollaborative **A**pplication **M**arkup **L**anguage und gehört zur Microsoft SharePoint-Technologiefamilie.

Als erstes wird ein SPQuery-Objekt mit dem Namen »abfrage« erzeugt. Dann folgt der String mit einem CAML-Ausdruck. Dieser ist wie folgt zu interpretieren:

»Suche nach einem Item, dessen Wert in der Spalte »Title« identisch ist mit dem Ausdruck »Termin 3«. Beachte, dass der Ausdruck ein String ist.«

Der CAML-Ausdruck wird dann in der folgenden Anweisung der Methode Query des SPQuery Objektes übergeben.

```
abfrage.Query = caml;
```

HINWEIS Sie sehen an der englischen Bezeichnung, dass der CAML-Ausdruck den internen Namen der Spalte benutzt. Warum dies so ist, wissen Sie. SharePoint-Anwender mit entsprechenden Berechtigungen können Spaltennamen, genauer den Anzeigenamen einer Spalte, verändern. Würde man also im CAML-Ausdruck diesen Namen verwenden, liefe eine Abfrage nach einer Änderung des Spaltennamens ins Leere.

Bis hierhin wurde definiert, dass es eine Query gibt und nach was zu suchen ist. Mit der Anweisung

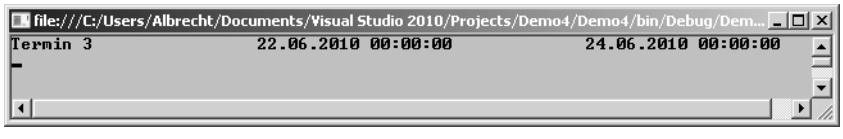
```
SPListItemCollection ergebnisse = kalender.GetItems(abfrage);
```

wird die eigentliche Suche gestartet und das Ergebnis steht in der ListItemCollection »ergebnisse« zur Verfügung.

Die Logik im weiteren Programmverlauf überprüft, ob ein entsprechendes Item gefunden werden konnte. Wenn nein, wird eine entsprechende Meldung auf der Konsole ausgegeben.

Wenn die Suche erfolgreich war, dann wird der Termin mit Anfangs- und Endezeit angezeigt. Da man nicht wissen kann, wie viele Treffer die Suche ergibt, wird die ListItemCollection in einer Schleife abgearbeitet. Das Ergebnis sehen Sie in der Abbildg. 26.21Abbildung 26.21 .

Abbildg. 26.21 Ausgabe des Projekts Demo5



Im Programmcode sehen Sie auch einen auskommentierten CAML-Ausdruck.

```
//string caml = "<Where><Eq><FieldRef Name='EventDate'/><Value Type='DateTime'>2010-06-
22T10:00:00Z</Value></Eq></Where>";
```

Hier würde nach allen Items gesucht werden, deren Feld »EventDate« dem 22.06.2010 entspricht. Man kann erkennen, dass der Wert eines Feldes typabhängig ist. Für eine Datumsabfrage, die einem

DateTime-Objekt von .NET Framework entspricht, muss ein entsprechendes internationales Format benutzt werden.

HINWEIS Wenn Sie tiefer in die SharePoint-Entwicklung einsteigen möchten, dann müssen Sie sich mit Datentypen und der Konvertierung von Datentypen vertraut machen.

Schöner wäre es natürlich, wenn man nach beliebigen Terminen suchen könnte. Dazu müssen Sie Befehlszeileneingaben auswerten zu können, um sie dann in die CAML-Zeichenkette einzubauen. Wie Sie das Programm entsprechend anpassen können, zeigt die Erweiterung unseres Beispiels aus Listing 26.8Listing 26.8.

Listing 26.8 Suchfunktion mit Konsoleeingabe

```
// .. using nicht mit abgedruckt
namespace Demo5
{
    class Program
    {
        static void Main(string[] args)
        {
            string termin = "Termin 3";
            string url = "http://localhost";
            string subWeb = "KalenderSite";
            string kalenderName = "Meinetermine";


            if (args.Count() >0)
            {
                termin = args[0];
            }
            SPSite sitecollection = new SPSite(url);

            SPWeb web = sitecollection.RootWeb;
            SPWeb kalenderWeb = web.Webs[subWeb];
            SPList kalender = kalenderWeb.Lists[kalenderName];

            // Abfrage erzeugen und starten
            SPQuery abfrage = new SPQuery();
            string caml = "<Where><Eq><FieldRef Name='Title'/><Value Type='Text'>"
                        + termin + "</Value></Eq></Where>";

            abfrage.Query = caml;

            // Ergebnis speichern und Auswerten
            SPListItemCollection ergebnisse = kalender.GetItems(abfrage);
            // wenn wir nichts gefunden haben
            if (ergebnisse.Count < 1)
            {
                Console.WriteLine("Es wurde kein Termin mit der Bezeichnung "
                                  + termin + " gefunden");
                Console.WriteLine("Zum beenden die Entertaste drücken");
                Console.ReadLine();
                return;
            }
            // alle gefundenen SPitems anzeigen
            foreach (SPItem item in ergebnisse)
```

Listing 26.8 Suchfunktion mit Konsoleeingabe *(Fortsetzung)*

```
            {

                Console.WriteLine(item["Titel"] + "\t\t" + item["Anfangszeit"]
                                                 + "\t\t" + item["Endzeit"]);
            }
            Console.WriteLine("Zum beenden die Entertaste drücken");
            Console.ReadLine();
        }
    }
}
```

Wichtig ist für Sie zu wissen, dass Parameter wie z.B. die Angabe des Termins, im Zeichenketten-Array »args[]« übergeben werden.

HINWEIS Ein Array ist eine Datenstruktur, in der Objekte vom gleichen Typ, in unserem Beispiel also Strings, abgelegt werden. Auf diese Objekte kann man dann mit ihrem Index zugreifen. Bei einem sogenannten »Assoziativen Array« haben die einzelnen Objekte einen Namen. Damit ist der Zugriff auch über den Namen möglich wie z.B. mit »item["Titel"]«. Hier wird auf ein Feld von »Item« zugegriffen, das den Namen »Titel« hat.

Der Ausdruck

```
args.Count > 0
```

ist dann wahr, wenn mindestens ein Parameter übergeben wurde. Wenn das der Fall ist, dann wird der erste Parameter

```
args[0]
```

in den String »termin« geschrieben.

HINWEIS Die Nummerierung, technisch formuliert der Index, eines Arrays beginnt in C# mit »0«. »Index 0« bezieht sich also auf den ersten Wert im Array »args«.

Die Zeile

```
string caml = "<Where><Eq><FieldRef Name='Title'/><Value Type='Text'>"
+ termin + "</Value></Eq></Where>";
```

ist nun »parametrisiert«. Das heißt, dass der Ausdruck variiert und zwar abhängig davon, was in der Variablen »termin« steht.

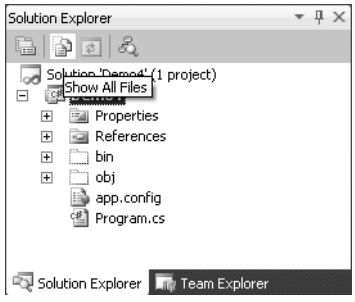
Experimentieren Sie mit dem Quellcode. Wenn Sie tiefer in die Abfrageprogrammierung einsteigen wollen, dann müssen Sie sich intensiv mit CAML als XML-Sprache und vor allem mit der Abbildung der unterschiedlichen Datentypen auseinander setzen. Dies gilt, wie bereits erwähnt, insbesondere für die Abfrage von Datumswerten.

HINWEIS CAML ist nicht die einzige Möglichkeit, um in SharePoint Abfragen zu programmieren. Eine Alternative ist LINQ, für **L**anguage **In**tegrated **Q**uery. LINQ ist eine Komponente von Microsoft .NET Framework zur Abfrage von Datenquellen. Mehr Informationen zu LINQ lesen Sie im Abschnitt »Silverlight« weiter hinten in diesem Kapitel.

Um unser Programm zu testen, müssen Sie es übersetzen und dann in einer Konsole starten. Gehen Sie wie in den folgenden Schritten beschrieben vor:

1. Klicken Sie im Solution-Explorer, wie in Abbildg. 26.22Abbildung 26.22 dargestellt, das Icon *Show All File*. Jetzt wird auch ein Ordner *bin* eingeblendet.

Abbildg. 26.22 Alle Dateien im Solution Explorer anzeigen



2. Öffnen Sie den Ordner und dann den Ordner *Debug*.
3. Öffnen Sie das Kontextmenü des Ordners Debug und wählen Sie die Option *Open Folder in Windows Explorer*.

Jetzt können Sie den Pfad zum Debug-Verzeichnis sehen, in dem sich die Datei *Demo5.exe* befindet.

4. Öffnen Sie jetzt eine Eingabekonsole über *Start/Ausführen* und den Befehl *cmd*
5. Wechsel Sie in das »Debug«-Verzeichnis und geben Sie den in Abbildg. 26.23Abbildung 26.23 dargestellten Befehl »demo5 "Termin 3"« ein.

Abbildg. 26.23 Programm mit Parametereingabe

Als Ergebnis wird der Termin gefunden und angezeigt. Es fehlt jetzt nur noch eine Funktionalität, deren Programmierung Sie noch nicht kennen gelernt haben. Das ist das Löschen eines Elementes.

Sechste Anwendung – ein Listenelement suchen und löschen

So gehen Sie vor um Listenelemente zu suchen und zu löschen:

Legen Sie wie gewohnt ein neues Projekt mit dem Namen *Demo6* als Konsolenanwendung an.

Der Aufbau der Anwendung entspricht im Wesentlichen dem von *Demo5* (s. Abschnitt »Fünfte Anwendung – ein Listenelement suchen und anzeigen«). Der Unterschied besteht darin, dass eine

andere Methode auf das gefundene Item angewendet und anders auf die Itemsammlung zugreifen wird.

Im folgenden Listing sehen Sie deshalb nur den für diese Unterschiede relevanten Teil des Quellcodes. Geben Sie den in Listing 26.9Listing 26.9 dargestellten zusätzlichen Quellcode ein.

Listing 26.9 Listenelement suchen und löschen

```
            // Ergebnis speichern und Auswerten
            SPListItemCollection ergebnisse = kalender.GetItems(abfrage);
if (ergebnisse.Count < 1)
            {
                Console.WriteLine("Es wurde kein Termin mit der Bezeichnung " + termin
                                  + " gefunden");
                Console.WriteLine("Zum beenden die Entertaste drücken");
                Console.ReadLine();
                return;
            }
            // das erste gefundene SPItems löschen
            ergebnisse[0].Delete();
            Console.WriteLine("Item wurde gelöscht");
            Console.WriteLine("Zum beenden die Entertaste drücken");
            Console.ReadLine();
```

Sie können erkennen, dass das Suchergebnis jetzt nicht mehr in einer Schleife abgearbeitet wird, sondern der erste Treffer direkt gelöscht wird.

Dazu wird die entsprechende Methode »Delete()« von SPItem genutzt.

1. Öffnen Sie wieder ein Konsolenfenster.
2. Starten Sie die Anwendung und geben Sie als ersten Parameter den Termin »Termin 1« ein.

Die Ausgabe der Anwendung sieht nach dem ersten Starten entsprechend Abbildg. 26.24Abbildung 26.24 aus.

Abbildg. 26.24 Ergebnis von *Demo6* bei gefundenem Termin

Wenn Sie die Anwendung nochmals starten und dabei den gleichen Termin suchen, erhalten Sie die in Abbildg. 26.25Abbildung 26.25 dargestellte Ausgabe – ein Beweis dafür, dass der Listeneintrag, das Item, tatsächlich gelöscht wurde.

Abbildg. 26.25 Ergebnis von *Demo6* bei nicht gefundenem Termin

Damit ist die grundlegende Einführung in die SharePoint-Programmierung abgeschlossen. Im folgenden Abschnitt wird gezeigt, wie Sie Anwendungen auf einem beliebigen Client, z.B. einem Arbeitsplatz programmieren können.

Client-Objektmodell

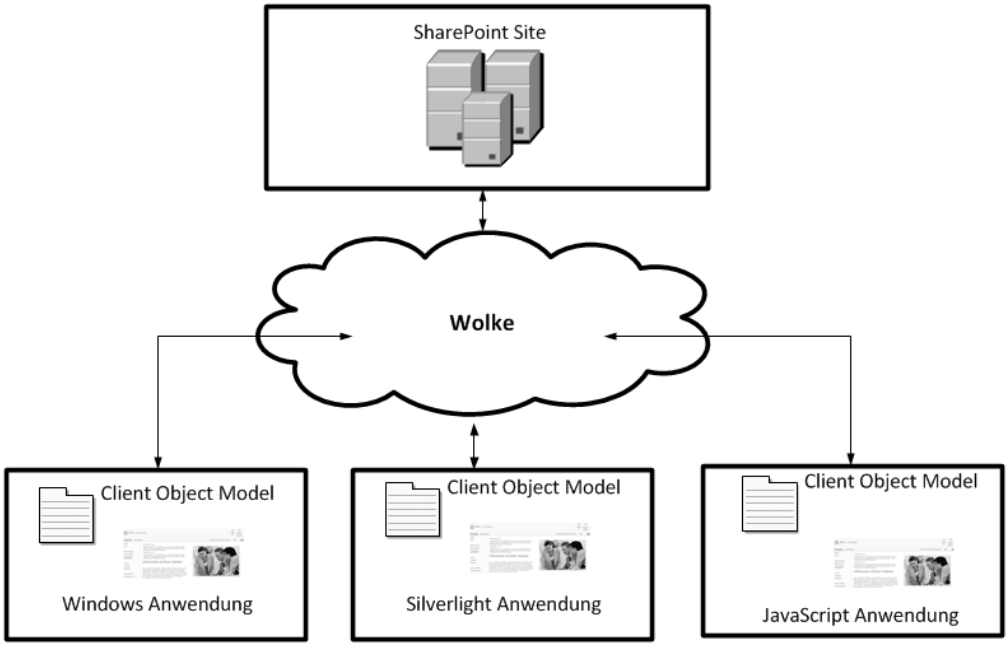
Eine wesentliche Erleichterung der Arbeit für den Entwickler stellt das in SharePoint 2010 neue Client-Objektmodell dar. Gerade im Hinblick auf die Entwicklung von Silverlight-Webapplikationen in Verbindung mit SharePoint, wird der Entwickler deutlich entlastet.

Wie der Namen schon andeutet, können die zum Client-Objektmodell gehörenden Klassen dazu verwendet werden, auf einem Clientsystem wie Windows 7, SharePoint-Anwendungen zu entwickeln und auch auszuführen.

Mit Hilfe des Client-Objektmodells stehen fast alle Eigenschaften und Daten einer Websitesammlung zur Verfügung, so zum Beispiel der Zugriff auf alle Objekte, die Sie bisher auch im Server Objektmodell kennen gelernt haben. Das Client-Objektmodell ist damit eine Alternative zu den Web Services, die wesentlich unkomfortabler zu programmieren sind.

Die folgende Grafik in Abbildg. 26.26Abbildung 26.26 zeigt die Architektur des Client-Objektmodells.

Abbildg. 26.26 Architektur des Client-Objektmodells



Das Modell soll am Beispiel unseres ersten Anwendungsfalls demonstriert werden – dem Auslesen der Listen einer Website. Sowohl die Vorgehensweise als auch der Quellcode werden ausführlich beschrieben.

Versuchen Sie, auch die anderen Anwendungsfälle auf der Basis des Client-Objektmodells umzusetzen. Sie wissen schon aus der Gegenüberstellung der beiden Objektmodelle, dass es zu jedem Serverobjekt auch ein entsprechendes Clientobjekt gibt. Serverseitig werden

- SPSite
- SPWeb
- SPList

verwendet.

Jetzt werden die Objekte

- Site
- Web
- Item

des Client-Objektmodells verwendet.

Starten Sie das erste Projekt!

1. Legen Sie zunächst ein neues Projekt mit dem Namen *Demo7* als Konsolenanwendung an. (s. Seite 842).
2. Jetzt müssen Sie darauf achten, die richtige Klassenbibliothek – das richtige Objektmodell – einzubinden. In unserem Beispiel ist das die Datei Microsoft.SharePoint.Client.dll.

3. Als weitere Assembly binden Sie die Datei Microsoft.SharePoint.Client.Runtime.dll ein.

Die zweite Bibliothek ist von entscheidender Bedeutung. Sie repräsentiert sozusagen den Server auf der Clientseite. Ohne das Einbinden dieser Datei wäre der Zugriff auf den Server nicht möglich. Warum dies so ist, erfahren Sie bei der Analyse des Beispielprogramms. Sie finden die Dateien in demselben Verzeichnis, aus dem Sie auch das Servermodell geladen haben.

Im Folgenden sehen Sie die Umsetzung unseres ersten Beispiels. Ein erster Blick zeigt schon wesentliche Unterschiede.

Listing 26.10 Zugriff auf SharePoint-Listen über das Client Objektmodell

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Microsoft.SharePoint.Client;


namespace Demo6
{
    class Program
    {
        static void Main(string[] args)
        {

            ClientContext context = new ClientContext("http://localhost");

            Site siteCollection = context.Site;
            context.Load(siteCollection);

            Web web = siteCollection.RootWeb;
            context.Load(web);

            ListCollection alleListe = web.Lists;
            context.Load(alleListe);

            context.ExecuteQuery();

            foreach (List list in alleListe)
            {
                Console.WriteLine(list.Title);
            }
            Console.ReadLine();
        }
    }
}
```

- Der Zugriff auf den entfernten Server erfolgt über die »ClientContext«-Klasse.
- Diese wiederum ist von der Klasse »ClientRuntimeContext« abgeleitet.
- Die Klasse befindet sich aber nicht im Client-Objektmodell sondern im Assembly *Microsoft.SharePoint.Client.Runtime*.

Das ist der Grund, warum Sie zwei Assemblys einbinden müssen.

Nach der Erzeugung einer Instanz des »ClientContext«, hier »context« genannt, kann man auf den entfernten SharePoint Server zugreifen. Damit wird auch deutlich, warum als Parameter die Adresse des Server bzw. der Websitesammlung, angeben werden muss.

Erst mit dem Aufruf der Methode »ExecuteQuery()« wird ein Datenpaket an den entfernten SharePoint gesendet. Alle Anweisungen, die Sie davor sehen, legen nur fest, was auf dem Server geschehen soll.

Die Methode »Load()« des Context lädt also nicht die Websitesammlung oder die Site sondern sagt nur, dass sie später auf dem Server geladen werden soll. Damit ist das Programm so zu interpretieren:

Es wird eine Verbindung zum Server aufgebaut. Danach wird in der Zeile

```
Site siteCollection = context.Site;
```

eine Instanz von Site erzeugt, die »siteCollection« heißt.

Für das Verständnis ist es nun äußerst wichtig zu erkennen, dass mit dieser Zeile keineswegs auch tatsächlich die Site zur Verfügung steht.

Die folgende Zeile

```
context.Load(siteCollection);
```

lädt auch nicht etwa die Site sondern legt nur fest, dass Sie später auf dem Server geladen werden soll. Der gleiche Zusammenhang gilt dann auch für die folgenden Anweisungen.

Erst nach Aufruf in der Zeile

```
context.ExecuteQuery();
```

werden alle vorher definierten Anweisungen auf dem Server ausgeführt und das Ergebnis an den Client gesendet. Mit anderen Worten, der gesamte Ablauf erfolgt auf dem Server und das Ergebnis wird über die Schleife auf dem Client angezeigt.

Das Ergebnis nach dem Aufruf des Programms entspricht dem von Demo1 im Abschnitt »Erste Anwendung – Listen einer Website anzeigen«.

Der deutlichste Unterschied im Vergleich zum serverseitigen Objektmodell liegt also in der Zugriffszeit auf den SharePoint. Während im Server-Objektmodell die Daten direkt in die Objekte geladen und weiterverarbeitet werden können, werden im clientseitigen Objektmodell die Daten erst mit der Ausführung von »ExecuteQuery()« den internen Objekten zur Verfügung gestellt.

Dieses Verfahren ermöglicht es dem Entwickler, mehrere Objektaufrufe in einer Serveranfrage zu verpacken, und damit die Verzögerung, die durch den Netzzugriff erfolgt, zu minimieren.

Im Prinzip wird hier die Programmierung eines Web Service gekapselt. Die Datenübertragung erfolgt aber auf der Basis eines Datenprotokolls, das »JSON« heißt. JSON ist das Akronym für **J**ava**S**cript **O**bject **N**otation.

Im Rahmen dieser Einführung in die SharePoint-Programmierung sind vor allem zwei Dinge wichtig:

- Zum einen können mit JSON Objekte im Textformat übertragen werden.
- Zum anderen ist JSON ein schlankes Format. Der zu übertragende Overhead an Steuerinformationen ist wesentlich geringer als bei Web Services, die auf SOAP als XML-Dialekt aufgebaut sind.

Dazu aber mehr im folgenden Abschnitt.

SharePoint Web Services programmieren

Bisher haben Sie zwei Programmiermodelle kennen gelernt – die Programmierung auf dem Server auf der Basis des Server-Objektmodells und die Programmierung auf dem Client mit Netzwerkzugriffen über das Client-Objektmodell.

Web Services stellen ein weiteres Programmiermodell zur Verfügung und sind eine Alternative zum Client-Objektmodell. Das heißt, Programme, die über ein Netzwerk auf den SharePoint zugreifen wollten, mussten bis einschließlich SharePoint Server 2007 mit Web Services programmiert werden.

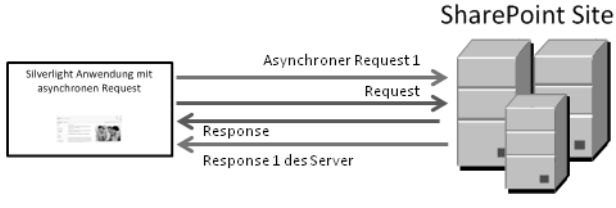
Ein Einstieg in die Web Services würde den Rahmen dieses Buches sprengen. Deshalb möchte ich nur die wichtigsten Details aufführen:

- SharePoint stellt für den Zugriff auf alle Elemente Web Services zur Verfügung.
- Web Services greifen immer über ein Netzwerkprotokoll auf den Server zu. In der Regel ist das das HTTP-Protokoll.
- Web Services übertragen XML-Daten als Befehle und Ergebnisse. Das verwendete Protokoll heißt SOAP, **S**imple **O**bject **A**ccess **P**rotocol. SOAP hat den Nachteil, dass sehr viele Steuerinformationen als Overhead verwendet werden. Der Anteil der tatsächlich transportierten Nutzdaten ist relativ gering. Aus Sicht der Geschwindigkeit ist JSON das bessere Format.
- Web Services brauchen clientseitig eine Repräsentation des Servers. Dieser Client wird häufig als Proxy oder SOAP-Client bezeichnet.
- Der Aufruf von Web Service-Funktionen ist in der Regel asynchron. Das heißt, die Anwendung wird nicht blockiert. Als Entwickler muss man Mechanismen schaffen, die erkennen, dass eine Antwort des Servers empfangen wurde. Die hier verwendete Technik heißt Callback Handler.

Mit dieser kleinen Aufzählung wird schon deutlich, wie sehr sich Web Services und das Client-Objektmodell ähneln. Praktisch ist es so, dass mit SharePoint 2010 das Client-Objektmodell eine Alternative zu Web Services darstellt, die wesentlich komplexer und umständlicher zu programmieren sind.

Die folgende Abbildg. 26.27Abbildung 26.27 zeigt die allgemeine Architektur eines Web Service.

Abbildg. 26.27 Architektur eines Web Services mit SharePoint 2010



Die Grafik zeigt, dass Web Service-Anfragen (Requests) asynchron verarbeitet werden. Auf eine Anfrage kann die Anwendung eine weitere Anfrage senden ohne auf die Antwort des Servers warten zu müssen. Das heißt aber auch, dass es einen Mechanismus geben muss, der die Antworten (Response) erkennt und die empfangenen Daten entsprechend verarbeitet.

HINWEIS Die Programmierung eines Web Services wird ausführlich im Abschnitt »Silverlight« beschreiben. Silverlight-Zugriffe auf den Server erfolgen immer über Web Services.

Programmierung von SharePoint-Webparts

Webparts sind Bausteine des SharePoint, die in eine ASPX-Seite eingebunden werden können. Damit sind auch Listen und Dokumentenbibliotheken Webparts. Webparts können Daten aus anderen Datenquellen, Inhalte von Listen, Suchergebnisse, Formulare und andere Webseiten darstellen. Der Grundgedanke ist, dass SharePoint-Seiten aus beliebigen Webparts immer wieder neu zusammengesetzt werden können.

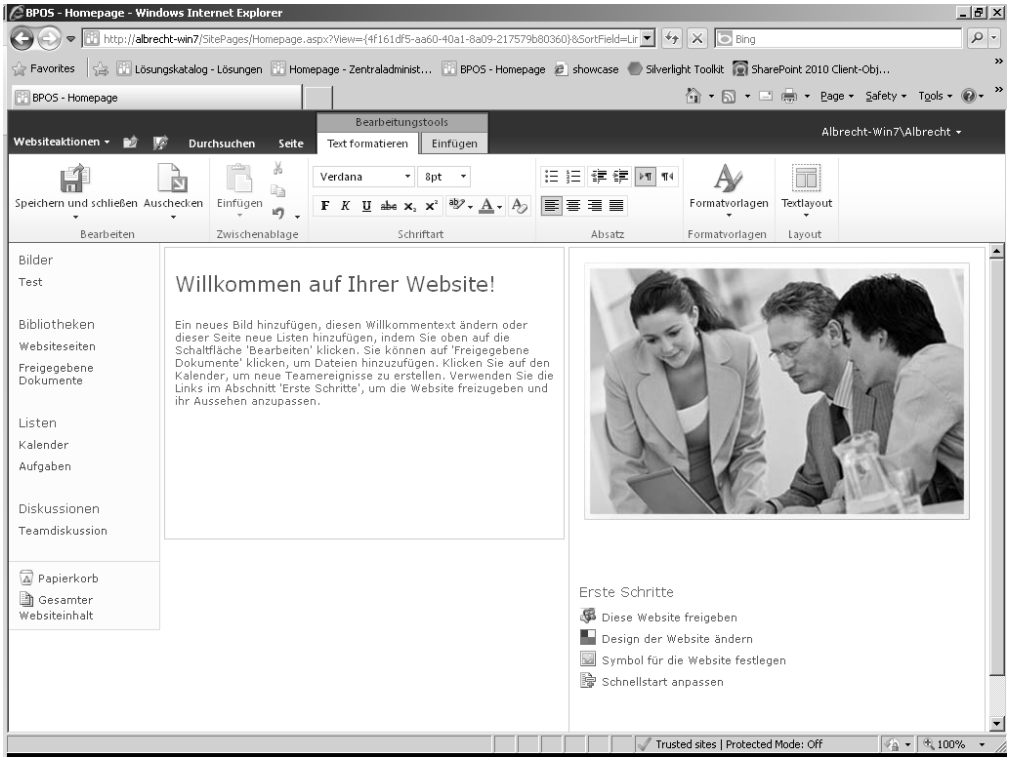
In diesem Kapitel beschränken wir uns auf den Bereich der visuellen Webpart-Programmierung. SharePoint unterscheidet zwischen visuellen Webparts und einfachen Webparts. Visuelle Webparts sind Bestandteile der grafischen Benutzeroberfläche, der sogenannten GUI, **G**raphical **U**ser **I**nterface. Zu den einfachen Webparts gehören Listen und Dokumentenbibliotheken. SharePoint 2010 stellt schon eine Vielzahl von Webparts zur Verfügung. Webparts, die Sie selbst programmieren, werden als Custom Webparts bezeichnet.

In den folgenden Absätzen lesen Sie, wie Sie Schritt für Schritt die Programmierung eines visuellen Webparts vornehmen. Die Abbildg. 26.28Abbildung 26.28 zeigt die Startseite der Website vor dem Einfügen des noch zu entwickelnden Webparts.

Die Entwicklung umfasst folgende Schritte:

1. Definition des Anwendungsfalls, also: Für was soll das Webpart verwendet werden?
2. Design des Webparts mit Definition der Komponenten, Controls, wie Eingabefelder, Listen, Kalender, Auto Complete Felder und vieles mehr.
3. Erstellung eines Visuell Webpart-Projekts in Visual Studio.
4. Programmierung
5. Bereitstellung (Deployment)
6. Einbinden in eine SharePoint-Seite.

Abbildg. 26.28 Startseite ohne Custom Webpart



Definition Anwendungsfall

Das zu entwickelnde Webpart soll dem Benutzer die Möglichkeit bieten, aus einem Kalender ein Datum auszuwählen, das dann in ein Eingabefeld geschrieben wird. In einem Textfeld wird dann die Differenz zwischen dem aktuellem Datum und dem ausgewählten Datum in Tagen angegeben.

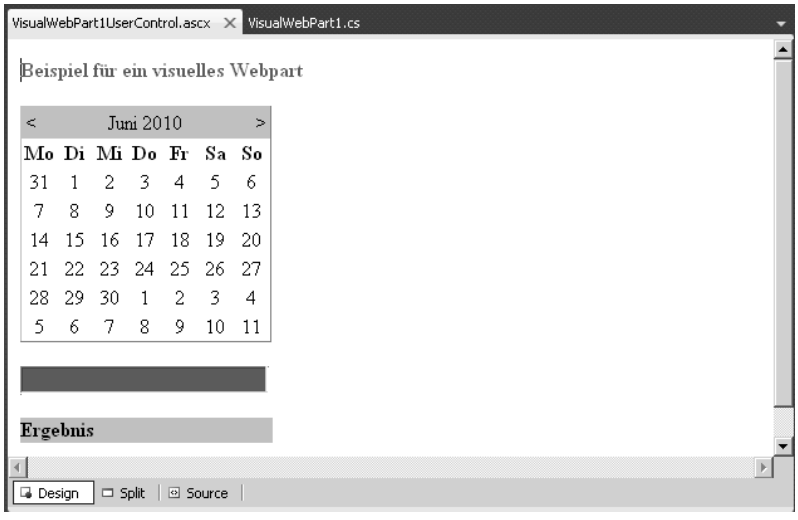
Dieser Anwendungsfall bietet zwar keinen besonderen Nutzen, zeigt aber die wichtigsten Techniken wie

- Einsatz einer fertigen Komponente
- Reaktion auf Mausereignisse
- Schreiben in Steuerelemente
- Durchführen von Berechnungen
- Formatierung von Steuerelementen

Design

Abbildg. 26.29Abbildung 26.29 zeigt das Design des Webparts im Visual Studio Editor. Hier sehen Sie die Verwendung der Editoransicht »Design«.

Abbildg. 26.29 Design des visuellen Webparts in der *Design*-Ansicht des Visual Studio Editors



In diesem Design werden die folgenden Steuerelemente verwendet:

- Ein Label, welches als Überschrift dient
- Ein Kalender
- Eine »TextBox« für die Ausgabe des ausgewählten Datums oder die manuelle Eingabe eines Datums
- Ein weiteres Label für die Ausgabe der Differenz in Tagen

Im folgenden Abschnitt zeigen wir Ihnen, wie Sie Oberfläche zusammenbauen und die Interaktion der einzelnen Komponenten untereinander programmieren können.

Erstellen eines Visual Webpart-Projekts

Starten Sie Ihr Projekt:

1. Öffnen Sie Visual Studio und wählen Sie den Menübefehl *File/New/Project*.
2. Hier finden Sie im Knoten *SharePoint/2010* die Vorlage Visuelles Webpart.
3. Wählen Sie diese Vorlage aus und vergeben Sie einen Namen für das Projekt.

TIPP Da Webparts als Komponenten in vielen Projekten verwendet werden können, sollten Sie eine Namenskonvention verwenden, die eindeutig ist. Die empfohlene Vorgehensweise besteht darin, die URL Ihres Unternehmens zu verwenden, da eine URL nur einmal vergeben wird. Dabei wird die Domänenreihenfolge getauscht. Aus »meinefirma.de« wird »de.meinefirma«.

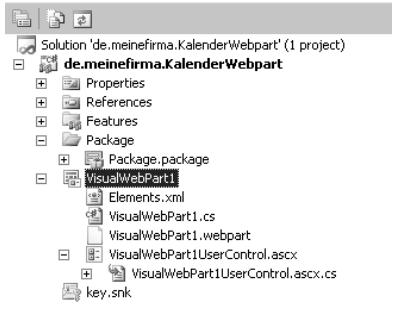
Ein möglicher Projektname wäre also »de.meinefirma.MeinWebpart«.

In unserem Beispielprojekt entscheiden wir uns für »de.meinefirma.KalenderWebpart«.

4. Nach Eingabe des Namens klicken Sie auf OK. Damit übernehmen Sie auch alle Standardeinstellungen.
5. Bestätigen Sie die Verbindungsaufforderung zu Ihrem Entwicklungs-SharePoint.

Die Abbildg. 26.30Abbildung 26.30 zeigt den Solution Explorer des in Visual Studio geöffneten Projekts.

Abbildg. 26.30 Das Projekt *de.meinefirma.KalenderWebpart*



6. Sollten Sie die Datei VisualWebPart1.cs nicht schon im Editor geladen haben, dann öffnen Sie nun die Datei VisualWebPart1.cs im Solution Explorer.

Sie sehen im Editor dann den Quellcode zu Listing 26.11Listing 26.11.

Listing 26.11 Standardquellcode für ein visuelles 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 de.meinefirma.KalenderWebpart.VisualWebPart1
{
    [ToolboxItemAttribute(false)]
    public class VisualWebPart1 : WebPart
    {
// Der Pfad wird unter Umständen automatisch aktualisiert, wenn das Visual
// WebPart-Projektelement geändert wird.
private const string _ascxPath = @"~/_CONTROLTEMPLATES/de.meinefirma.KalenderWebpart/
VisualWebPart1/VisualWebPart1UserControl.ascx";

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

Listing 26.11 Standardquellcode für ein visuelles Webpart

```
    }
}
```

Im Folgenden beschreiben wir Ihnen den Aufbau des vorliegenden Quellcodes. Da dieser für alle Webparts der gleiche ist, lernen Sie hier die Grundlagen für eine spätere Vertiefung.

- Zu Beginn sehen Sie die Using-Anweisungen, die Komponentenbibliotheken einzubinden. Damit können Sie die in diesen Dateien vorhandenen Komponenten in Ihrem Webpart verwenden. Es handelt sich hier um Bibliotheken von .NET Framework und der SharePoint API.
- Die folgende Zeile definiert den Namespace des Webparts. Durch die verwendete Namenskonvention ist der Name des Webparts eindeutig.

```
namespace de.meinefirma.KalenderWebpart.VisualWebPart1
```

HINWEIS Generell kennzeichnen Namespaces Einheiten, die zusammengehören. So ist das gesamte .NET Framework, auf dessen Basis SharePoint-Webparts programmiert werden, in Namespaces strukturiert. Namespaces sind hierarchisch organisiert. Die einzelnen Ebenen sind durch die Punktnotation getrennt. Wir haben in unserem Beispiel folgende drei Namespaces:

- de
- meinefirma
- KalenderWebpart

- Die im folgenden definierte Klasse VisualWebPart1 gehört zum Namespace »KalenderWebpart«, welches zum Namespace »meinefirma« gehört, der wiederum Bestandteil des Namespace »de« ist.
- Die erste Zeile in der Definition des Namespace »de.meinefirma.KalenderWebpart.VisualWebPart1« ist ein Attribut. Attribute werden immer in eckigen Klammern notiert und stehen unmittelbar vor dem Element, welches Sie erweitern.

```
[ToolboxItemAttribute(false)]
```

HINWEIS Das hier verwendete Attribut verweist darauf, dass es sich bei dem definierten Webpart nicht um eine Komponente der Toolbox handelt. Für das Verständnis dieses Beispiels ist nur wichtig zu wissen, dass Attribute dazu da sind, weiteren Quellcode in das Programm einzubinden.

- Unsere eigentliche Klasse wird in der folgenden Zeile definiert:

```
public class VisualWebPart1 : WebPart
```

  Sie besagt, das »VisualWebPart1« von der Klasse »WebPart« abgeleitet ist. Die Klasse erbt damit alle Eigenschaften eines Webparts und kann nun entsprechend der Vorgaben durch den Anwendungsfall erweitert werden.

HINWEIS Erben heißt in diesem Zusammenhang, dass »VisualWebPart1« per se ein »WebPart« ist und damit auch schon alle Eigenschaften eines SharePoint Webparts besitzt. Es werden also lediglich weitere visuelle Komponenten zum WebPart hinzugefügt und logisch miteinander verknüpft. Das heißt aber auch für Einsteiger im Bereich der SharePoint-Programmierung, dass Sie die Klasse »WebPart« genau analysieren müssen, um komplexe Webparts zu programmieren.

TIPP Bevor Sie damit beginnen, Erweiterungen für Ihr visuelles Webpart zu programmieren, sollten Sie nachschauen, ob diese Erweiterung nicht schon von »WebPart« geerbt werden kann.

- In der nächsten Anweisung wird eine Zeichenkonstante definiert, die den Pfad Vorlage für Steuerelemente beinhaltet. Auf die hier verwiesene Datei werden wir noch im weiteren Verlauf des Kapitels eingehen. Im Moment ist es wichtig zu wissen, dass in dieser Datei die visuellen Komponenten beschrieben werden.
- Die Klasse besteht im Moment aus einer einzigen Funktion, »CreateChildControls()«, die genau das tut, was ihr Name andeutet, nämlich alle zusätzlichen Controls, also die Steuerelemente, des Webparts zu erzeugen. Und diese zusätzlichen visuellen Komponenten sind in der Datei beschrieben, auf die die Variable »_ascxPath« verweist.
- Die Funktion erzeugt ein Controlobjekt und »lädt« in dieses Control alle Controls, die in der oben erwähnten Datei beschrieben sind.

```
Control control = Page.LoadControl(_ascxPath);
```

  Diese zusätzlichen Controls werden dann in der zweiten Anweisung der Liste aller bereits vorhandenen Controls des Webparts hinzugefügt.

```
Controls.Add(control);
```

  Die entsprechende Eigenschaft, die »VisualWebPart1« geerbt hat, heißt sinnigerweise »Controls« und verfügt über die Methode »Add«.

Diese Vorgehensweise ist immer die gleiche. Als Entwickler eines eigenen Webparts definieren Sie die Komponenten, die Sie einem Standard-Webpart hinzufügen möchten. Der dafür notwendige Quellcode ist in der Vorlage »Visual Webpart« schon abgelegt. Natürlich können Sie auch ein leeres Projekt öffnen und die in Listing 26.11Listing 26.11 gezeigten Zeilen selber schreiben.

HINWEIS Webparts, die Sie selber schreiben, werden als *Custom Webparts* bezeichnet.

Die Beschreibung aller zu ladenden Controls steht, wie bereits beschrieben, in der Abbildg. 26.31Abbildung 26.31 gezeigten Datei »VisualWebPart1UserControl.ascx«.

Listing 26.12 Definition zusätzlicher Steuerelemente

```
<%@ Assembly Name="$SharePoint.Project.AssemblyFullName$" %>
<%@ Assembly Name="Microsoft.Web.CommandUI, Version=14.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c" %>
```

Listing 26.12 Definition zusätzlicher Steuerelemente *(Fortsetzung)*

```
<%@ 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="VisualWebPart1UserControl.ascx.cs"
Inherits="de.meinefirma.KalenderWebpart.VisualWebPart1.VisualWebPart1UserControl" %>
```

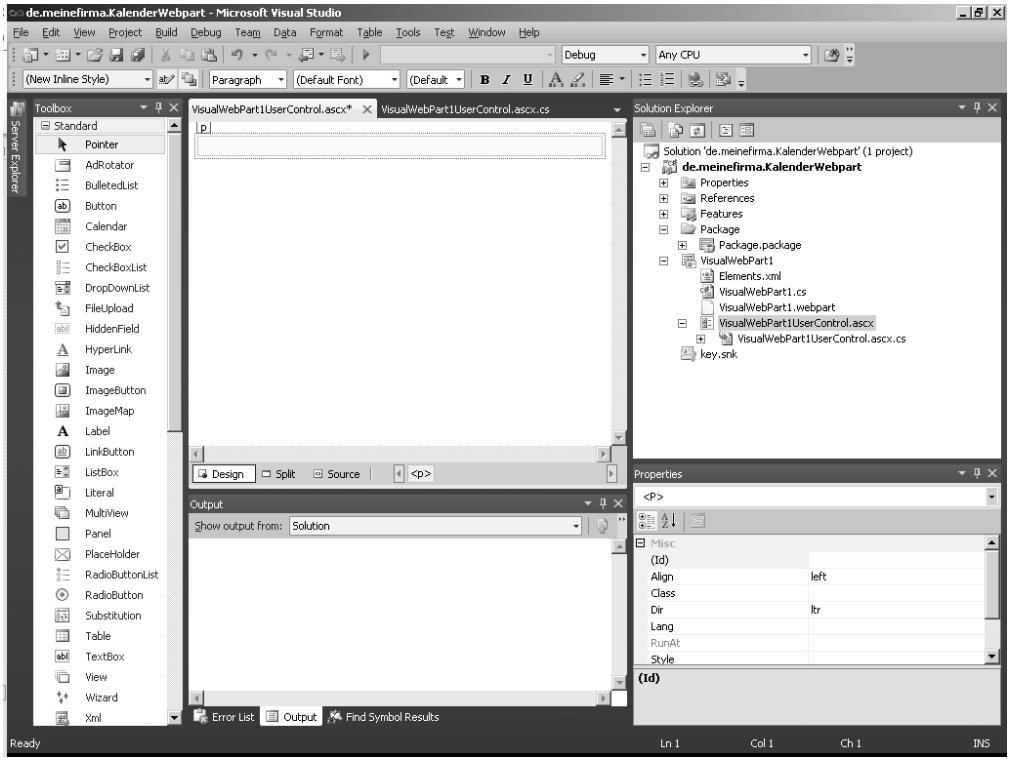
Für das Verständnis wichtig sind vor allem die letzten beiden Direktiven.

HINWEIS Eine Direktive erkennen Sie an der Notation <%@ %>.

Die Direktiven besagen, dass die Logik der hier noch zu beschreibenden Komponenten in einer eigenen Datei beschrieben wird, »VisualWebPart1UserControl.ascx.cs«, und dass die Datei die Controls des Webparts erbt.

Wenn Sie nun in Visual Studio in die Designansicht wechseln, dann werden Sie feststellen, dass Sie nichts sehen, denn aktuell sind noch keine visuellen Komponenten für das Webpart definiert.

Abbildg. 26.31 *VisualWebPart1UserControl.ascx* in der Designansicht

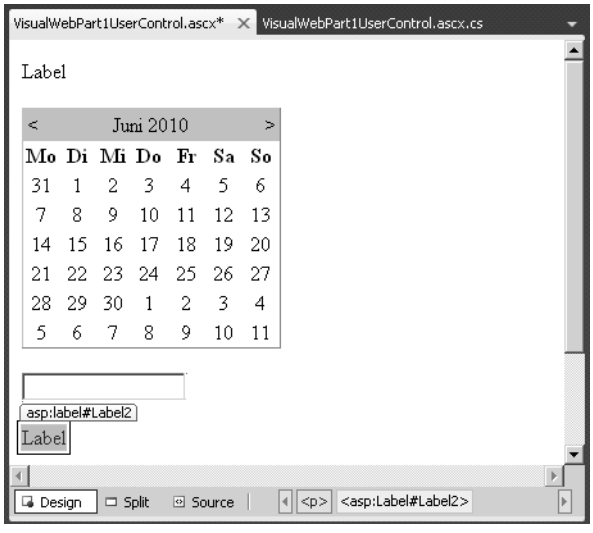


Sie sehen im linken Bereich von Visual Studio, dass die Toolbox nun mit Komponenten gefüllt ist.

1. Bleiben Sie in der Designansicht und ziehen Sie als erstes ein Label Control in den mit einer gepunkteten Linie umrahmten Bereich des Designfensters. Bei diesem Bereich handelt es sich um das HTML-Element *Paragraph*.
2. Drücken Sie dann ↵, um eine Leerzeile zu erzeugen.
3. Ziehen Sie als nächstes das Calender Control in den Entwurfsbereich.
4. Wiederholen Sie den Vorgang mit einer TextBox und einem weiteren Label. Drücken Sie zu Erzeugung von Leerzeilen zwischen den Komponenten jeweils die Taste ↵

In Abbildg. 26.32Abbildung 26.32 sehen Sie das Ergebnis Ihrer Aktion.

Abbildg. 26.32 Visuelle Komponenten in der Designansicht



5. Wechseln Sie nun in die Quellcodeansicht. Es sind die in Listing 26.13Listing 26.13 dargestellten Zeilen hinzugekommen.

Listing 26.13 ASP-Notation visueller Komponenten

```
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
<p>
    <asp:Calendar ID="Calendar1" runat="server"></asp:Calendar>
</p>
<p>
     </p>
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<br />
<p>
    <asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
</p>
```

6. Sie können jetzt in der Quellcodeansicht oder in der Designansicht den Text für die Labels und einige Formatierungsanweisungen eingeben. In unserem Projekt wurde das in der Quellcodeansicht umgesetzt.

Listing 26.14Listing 26.14 stellt den veränderten Quellcode dar.

Listing 26.14 Formatierte visuelle Komponenten

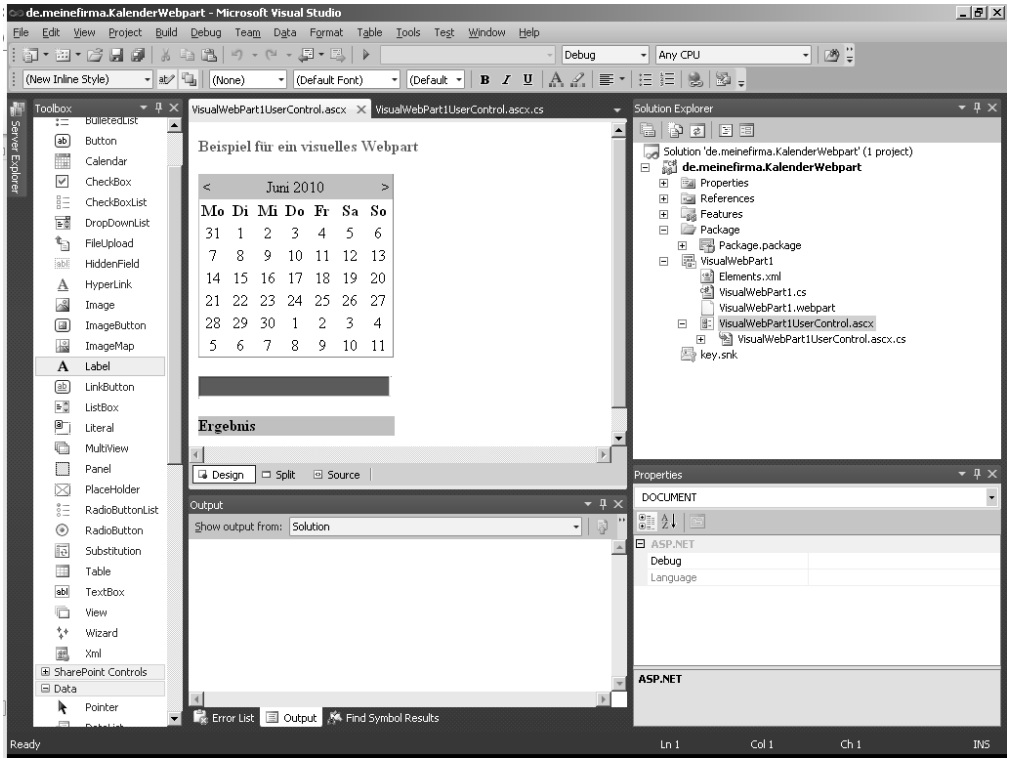
```
<asp:Label ID="Label1" runat="server" Text="Beispiel für ein visuelles Webpart"
    Font-Bold="True" ForeColor="#CC3300"></asp:Label>
<p>
    <asp:Calendar ID="Calendar1" runat="server" Width="200px"></asp:Calendar>
</p>
<asp:TextBox ID="TextBox1" runat="server" BackColor="#CC3300" Width="197px"
    Font-Bold="True" ForeColor="White"></asp:TextBox>
<br />
```

Listing 26.14 Formatierte visuelle Komponenten

```
<p>
    <asp:Label ID="Label2" runat="server" Text="Ergebnis" BackColor="Silver"
        Font-Bold="True" ForeColor="Black" Height="20px" Width="201px"></asp:Label>
</p>
```

Die Abbildg. 26.33Abbildung 26.33 zeigt das Ergebnis in der Designansicht.

Abbildg. 26.33 Formatierte visuelle Komponenten in der Designansicht



Damit ist der Entwurf der Benutzeroberfläche abgeschlossen. Jetzt geht es darum, die Logik einzubauen. In unserem Beispiel ist der Ausgangspunkt ein Ereignis, nämlich die Tatsache, dass der Anwender ein Datum ausgewählt hat.

Dazu ist es wichtig, alle beteiligten Komponenten benennen zu können. Das sind:

- der Kalender »Calender1«
- die TextBox »TextBox1«
- das Label »Label2«.

HINWEIS In der Praxis werden Sie natürlich sprechende Namen einer zuvor festgelegten Namenskonvention verwenden.

Nehmen Sie im Quellcode die folgende Ergänzung für die Kalenderkomponente vor.

```
<asp:Calendar ID="Calendar1" runat="server"
        onselectionchanged="Calendar1_SelectionChanged"></asp:Calendar>
```

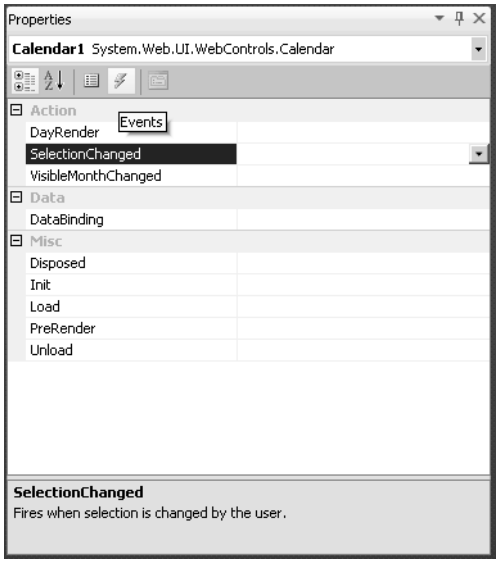
Damit definieren Sie ein Ereignis »onselectionchanged«, das immer dann auftritt, wenn der Benutzer einen Kalendereintrag ausgewählt hat.

Das Ereignis wird von einer Methode behandelt, die »Calender1_SelectionChanged« heißt. Die Methode ist ein sogenannter Event Handler und steht in der oben bereits referenzierten Datei VisualWebPart1UserControl.ascx.cs.

Sie können den Event Handler auch in der Entwurfsansicht zuweisen. So gehen Sie vor:

1. Öffnen Sie dazu das Kontextmenü des Calender Control oder markieren Sie es und drücken Sie Alt+↵.
2. Wählen Sie den Eintrag *Properties*.
3. Klicken Sie dann im Dialogfeld *Properties* auf Event und führen Sie einen Doppelklick auf die Aktion *SelectionChanged* aus.

Abbildg. 26.34 Event Handler in der Designansicht zuweisen



Wenn Sie jetzt die Datei VisualWebPart1UserControl.ascx.cs in Visual Studio öffnen, sehen Sie den Quellcode aus Listing Listing 26.1426.14.

Listing 26.15 Event Handler für den Kalender

```
using System;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

namespace de.meinefirma.KalenderWebpart.VisualWebPart1
{
    public partial class VisualWebPart1UserControl : UserControl
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }

        protected void Calendar1_SelectionChanged(object sender, EventArgs e)
        {

        }


    }
}
```

Jetzt muss man nur noch die Anweisungen schreiben, die den ausgewählten Kalendereintrag in die »TextBox« schreibt und anschließend die Differenz in Tagen berechnet. Dies wird dann im Label mit dem Namen »Label2« angezeigt.

4. Ergänzen Sie die Zeilen des nachfolgenden Listing 26.16Listings 26.16.

Listing 26.16 Quellcode für Event Handler mit Funktionalität

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
        {
            this.TextBox1.Text = this.Calendar1.SelectedDate.ToShortDateString();
            this.Label2.Text =  "Differenz in Tagen: " + (this.Calendar1.SelectedDate -
            DateTime.Now).Days;

        }
```

Damit haben Sie ein voll funktionsfähiges visuelles Webpart. Die nächsten Absätze beschreiben, wie dieses Webpart getestet und dann endgültig zur Verfügung gestellt werden kann.

HINWEIS In den Beispielen zur grundlegenden SharePoint-Programmierung haben Sie vor allem Klassen des SharePoint Objektmodells verwendet. In diesem Beispiel sehen Sie, dass man auch auf Klassen zugreifen kann, die prinzipiell in allen Windows-Anwendungen zum Einsatz kommen, z.B. das Calender-Objekt. Und das ist die große Stärke der visuellen Webparts. Sie können auf eine mächtige Programmbibliothek, das .NET Framework, zugreifen.

Webpart testen

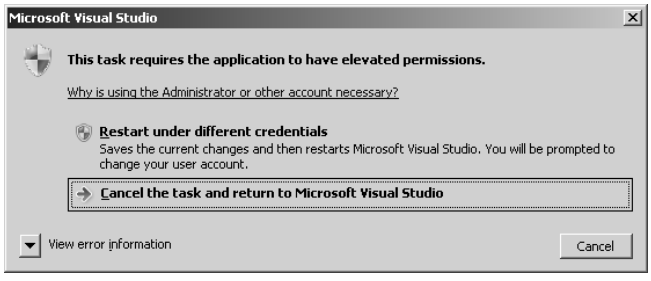
Das entwickelte Webpart kann nun getestet werden. Markieren Sie mit der Maus im Solution Explorer das Projekt und drücken Sie die Taste F5.

Jetzt laufen mehrere Prozesse ab:

- Die Anwendung wird kompiliert
- Es wird eine Webpart-Datei erstellt
- Visual Studio lädt die Datei auf den Server
- Visual Studio startet den Browser und öffnet die die Startseite der Website

HINWEIS Sollten Sie stattdessen die in Abbildg. 26.35Abbildung 26.35 dargestellte Fehlermeldung sehen, dann haben Sie Visual Studio nicht im Administratormodus gestartet.

Abbildg. 26.35 Fehlermeldung beim Bereitstellen eines Webparts



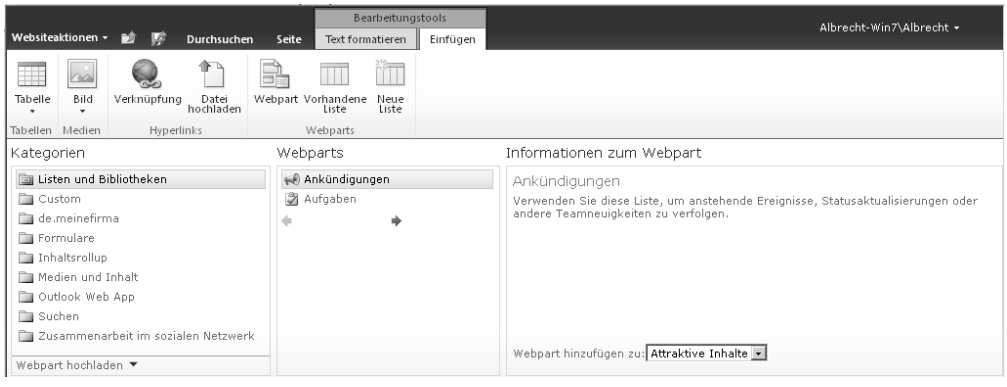
Starten Sie in diesem Fall Visual Studio wieder wie in Abbildg. 26.1Abbildung 26.1 gezeigt.

In den folgenden Schritten wird beschreiben, wie Sie das Webpart zu Testzwecken in die Startseite eingebunden werden kann.

1. Öffnen Sie Ihre SharePoint Seite und wählen Sie *Websiteaktionen/Seite bearbeiten*.
2. Klicken Sie auf *Einfügen/Webparts*.

Der Webpart-Bereich wird, entsprechend Abbildg. 26.36Abbildung 26.36, geöffnet.

Abbildg. 26.36 Dialogfeld zu Einfügen von Webparts



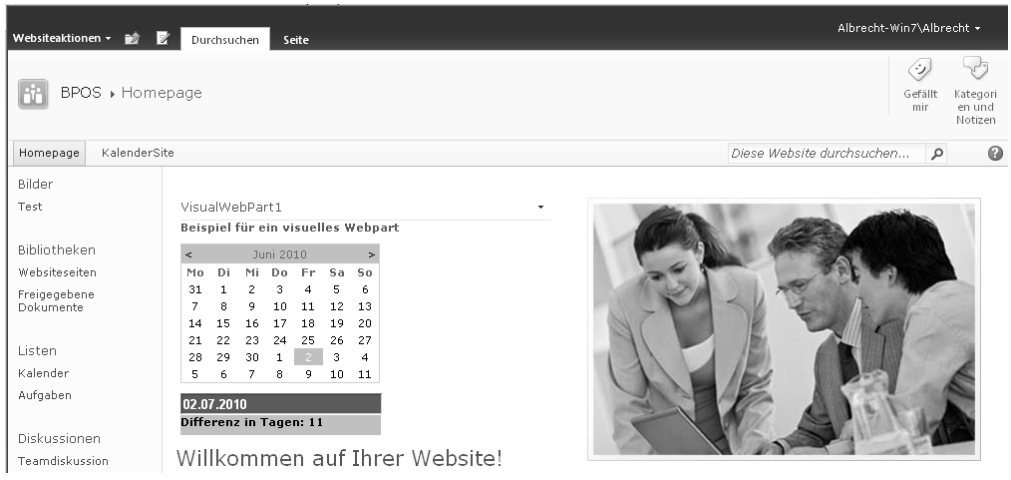
3. Wählen Sie in der Liste unter Kategorien den Eintrag *Custom*. Sie sehen alle Custom Webparts, also alle Webparts, die nicht Out-of-the-Box mit SharePoint mitgeliefert werden.
4. Wählen Sie jetzt im Bereich Webparts Ihr individuelles Webpart *VisualWebpart1* aus.
5. Klicken Sie auf die Schaltfläche Hinzufügen und beenden Sie die Eingabe durch Klicken auf Speichern und Schließen.

Das Webpart ist jetzt eingefügt und kann getestet werden.

- Klicken Sie im Kalender auf ein beliebiges Datum.
- In der TextBox wird das Datum angezeigt und im Label die berechnete Differenz.

Das Ergebnis stellt sich wie in Abbildg. 26.37Abbildung 26.37 dar.

Abbildg. 26.37 Getestetes Webpart



Beenden Sie den Test, indem Sie die Tastenkombination ⇧+F5 drücken. Visual Studio beendet damit das Debugging und schließt das Browserfenster.

Was jetzt noch fehlt, ist das dauerhafte zur Verfügung stellen des Webparts und seine Aktivierung auf einem Produktivsystem.

Bereitstellen von Webpart

Nach erfolgreichem Test kann ein Webpart auf einem Produktivsystem eingerichtet werden (Deployment).

HINWEIS Deployment bezeichnet den Vorgang des Verteilens einer Komponente oder Solution. Der Unterschied zu einer Installation über eine Setup-Routine besteht darin, dass beim Deployment die Komponenten in einen Container geladen werden, und das ist für unser Thema eine Website des SharePoint. Nach dem Laden werden die Komponenten dann aktiviert und stehen jetzt zur Verfügung.

ACHTUNG Das Deployment ist in der Regel die Aufgabe eines SharePoint-Administrators.

Mit dem Deployment ist dann der letzte Schritt in der Entwicklungskette abgeschlossen.

Die Sandbox – ein Überblick

Eine der größten Probleme für Entwickler unter SharePoint 2007 war die Tatsache, dass alle Solutions als vertrauenswürdig (trusted) galten. Farmadministratoren hatten deshalb die Sorge, dass Solutions der Stabilität des SharePoint schaden könnten. Die einzige Möglichkeit der Administratoren bestand darin, in den Quellcode zu schauen um möglichen schadhaften Code zu identifizieren. Die Lösung für dieses Problem heißt unter SharePoint 2010 Sandkasten-Lösung, bzw. »Sandboxed Solution«.

Sandboxed Solutions bieten Websitesammlungsadministratoren die Möglichkeit, Solutions bereitzustellen und zu überwachen. Diese Solutions sind im Prinzip nicht vertrauenswürdig aber sicher. Sicher heißt, dass im Extremfall nur die Websitesammlung, nicht aber die ganze Farm instabil werden kann.

HINWEIS Mit einer Sandbox (Sandkasten) bezeichnet man in der Softwareentwicklung einen Systembereich, in dem Software vom Rest des umgebenen Systems abgeschirmt ist. Die Software kann aus Systemsicht keinen Schaden anrichten. In der Regel gibt es zusätzlich noch Überwachungsmechanismen. Damit bietet die Sandbox, manchmal auch »Spielwiese« genannt, die Möglichkeit, Anwendungen zu nutzen, denen das Gesamtsystem nicht »traut« oder trauen darf. Typische Beispiele sind, neben der hier beschriebenen Sandboxed Solution unter SharePoint 2010, die Sandbox im Browser, die einer Anwendung z.B. den Zugriff auf lokale Ressourcen verweigert.

Unter SharePoint 2010 können Sandboxed Solutions überwacht und eingeschränkt werden. Die Einschränkung kann sich z.B. auf Ressourcen wie Arbeitsspeicher und CPU-Nutzung beziehen. Technisch geschieht dies durch mehrere Maßnahmen.

So laufen Sandboxed Solutions in einem eigenen, separaten Prozess, der durch die ».NET Code Access Security policy« in seinen Möglichkeiten eingeschränkt wird. Als Entwickler kann man bei der Programmierung einer solchen Solution nur einen Teilbereich der SharePoint API nutzen. Es stehen sozusagen nur Methoden zur Verfügung, von denen man annimmt, dass sie das Gesamtsystem nicht gefährden können. Die Überwachung der Solution übernimmt der SharePoint. Administratoren können entsprechende Einschränkungen, so genannte »Restrictions«, setzen.

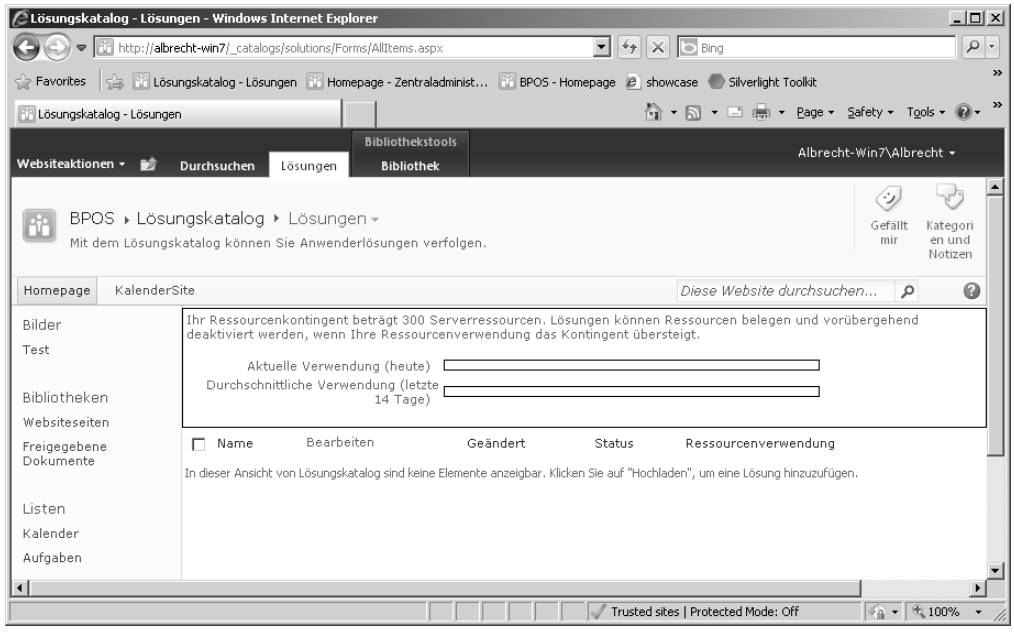
Der große Vorteil dieser Architektur besteht darin, dass der Entwicklungsprozess bis hin zum Deployment wesentlich flexibler und vor allem schneller gestalten werden kann. Man muss nicht beim Farmadministrator »nachfragen«, es genügt die Berechtigung als Websitesammlungsadministrator. Das ist natürlich auch im Zusammenhang mit den SharePoint Online Services von großem Vorteil. Hier besteht gar keine Möglichkeit, über die Websitesammlung hinaus administrativ tätig zu werden,

Selbst Solutions von Drittanbietern können jetzt schnell und ohne großen bürokratischen Aufwand installiert werden.

Farm Solutions sind demnach unter SharePoint 2010 »full trust solutions«. Alle SharePoint 2007 Solutions waren Farm Solutions und mussten deshalb auch von Farmadministratoren bereitgestellt werden. Sandboxed Solutions werden in einer neuen Solution Galerie verwaltet, die nichts anderes als eine Dokumentenbibliothek ist. Websitesammlungsadministratoren können hier Solutions hochladen. Das geschieht in gleicher Weise, wie in einer normalen Dokumentenbibliothek.

Zum Anzeigen dieser Bibliothek wählen Sie den SharePoint-Menübefehl *Websiteaktionen/Website-einstellungen/Lösungen*.

Abbildg. 26.38 Solution Galerie



Sie können in Abbildg. 26.38Abbildung 26.38 erkennen, dass in unserem Beispiel ein Kontingent von 300 Serverressourcen zur Verfügung steht und Sie sehen, dass Solutions vorübergehend deaktiviert werden, wenn die Ressourcenverwendung überschritten wird.

Administratoren können für Sandboxed Solutions sogenannte »Site Collection Quotas« definieren. Der Standard ist hier, entsprechend Abbildg. 26.38Abbildung 26.38, 300. Die Ressourcen selbst werden nach bestimmten Regeln berechnet. So entsprechen z.B. 20 SQL-Abfragen einem sogenannten »Ressource-Point«.

Wie Sie im Verlauf des Abschnitts noch sehen werden, kann Visual Studio Sandboxed Solution zur Entwicklungszeit hochladen und aktivieren.

HINWEIS Als Entwickler interessiert natürlich die Frage, welche Arten von Solution man in der Sandbox entwickeln kann. Die Frage kann schnell beantwortet werden: Man kann fast alle Anwendungsfälle umsetzen, solange diese sich im Gültigkeitsbereich einer Websitesammlung bewegen. Allerdings sind visuelle Webparts nicht erlaubt.

Sandbox-Klassen

Wie bereits erwähnt, ist die Sandbox-API eine Teilmenge der SharePoint API. Sandboxed Solutions-geeignete Klassen haben Sie schon kennen gelernt. Das sind nämlich unter anderem SPSite, SPWeb, SPList und SPListItem.

HINWEIS Eine vollständige Liste finden Sie in der Dokumentation zum SharePoint SDK (Software Development Kit).

Sie werden feststellen, dass die wesentlichen Klassen, die für Businessanwendungen benötigt werden, verwendet werden dürfen. Das sind unter anderem:

- Webparts
- Modules
- Lists
- Content Types
- Event Receivers
- Feature activation events
- Custom workflow actions
- InfoPath business logic

Die Grenze ist die Websitesammlung, in der die Solution zur Verfügung gestellt wird. Damit können Sie auf alle Listen und Dokumentenbibliotheken der Sammlung zugreifen. Allerdings werden auch die Einschränkungen deutlich. Zugriffe auf den Web Front End oder auf externe Daten via Web Services oder Datenbanken sind nicht möglich.

Das heißt aber nicht, dass generell der Zugriff auf externe Daten nicht möglich wäre. Sie können in der Sandboxed Solution die als sicher geltenden Business Connectivity Services (BCS) nutzen. Diese sind ein neues Features in Share Point 2010 und lösen den Geschäftsdatenkatalog (Business Data Catalog, BDC) unter SharePoint 2007 ab.

Die Regeln der Sandbox gelten nur für Code, der auf dem Server läuft. Silverlight und JavaScript können Web Services und andere Daten nutzen, wenn entsprechende Berechtigungen vergeben werden.

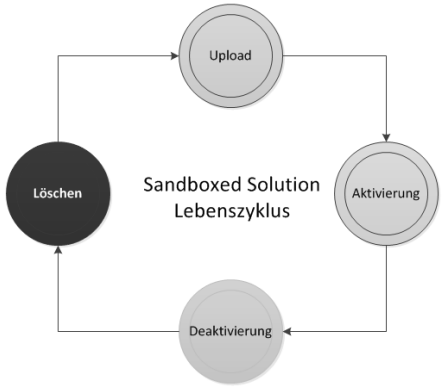
HINWEIS Eine solche Policy-Datei werden Sie im Kapitel »« kennen lernen.

Das heißt, dass Sie nahezu alle Features einer Farm Solution umsetzen können, wenn Sie einen Mix aus serverseitigem und clientseitigem Code einsetzen.

Sandboxed Solution durchlaufen vier Phasen eines Lebenszyklus. Dabei werden die typischen Phasen einer Farmsolution wie Hinzufügen, Deployment und Aktivierung in Upload und Aktivierung zusammengefasst.

Die Abbildg. 26.39Abbildung 26.39 zeigt den Lebenszyklus einer Sandboxed Solution.

Abbildg. 26.39 Lebenszyklus einer Sandboxed Solution



Ein Sandboxed Webpart

Das Programmieren eines Sandboxed Webparts unterscheidet sich nicht wesentlich von der Entwicklung einer Farm Solution. Im Folgenden werden wir die Vorgehensweise an einem Beispiel beschreiben. Es zeigt den Zugriff auf Listen.

Der Anwendungsfall lässt sich wie folgt beschreiben:

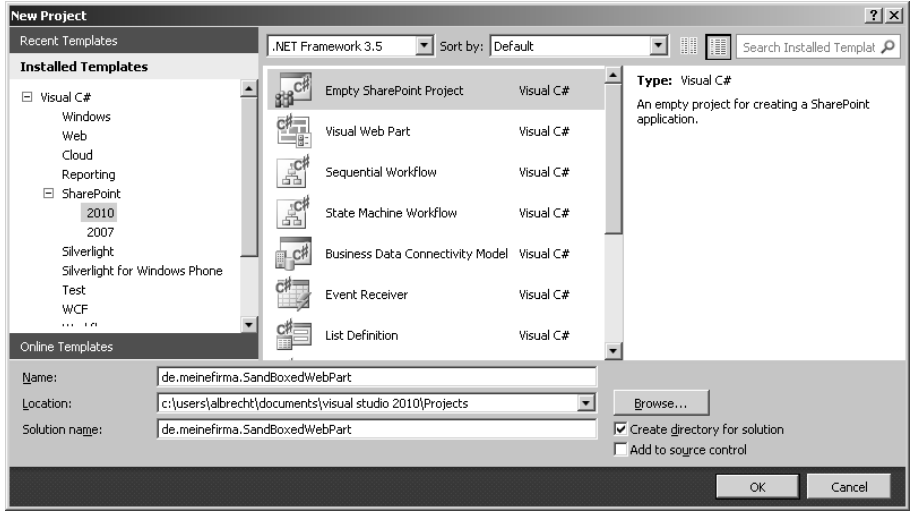
»Zeige alle Listen der Website an, in der die Sandboxed Solution bereitgestellt wurde.«

So gehen sie vor:

1. Öffnen Sie Visual Studio und legen Sie ein leeres SharePoint-Projekt mit dem Namen »de.meinefirma.SandBoxedWebPart« an.

HINWEIS Wie Sie ein neues Projekt in Visual Studio anlegen, lesen Sie auf Seite 842.

Abbildg. 26.40 Anlegen eines leeren SharePoint-Projekts

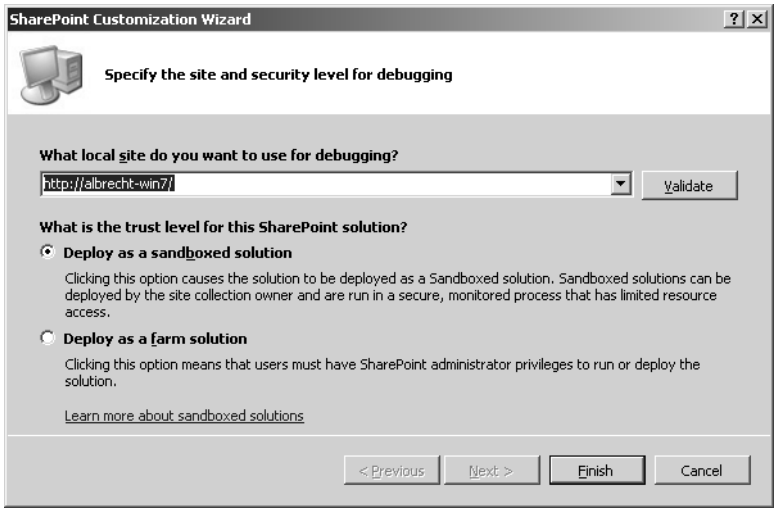


Sie müssen ein leeres SharePoint-Projekt anlegen, weil die Visual Webpart-Vorlage nicht unterstützt wird. Diese Vorlage beinhaltet, wie Sie aus unserem Grundlagenteil wissen, ein »ascx-Control«, welches in das Verzeichnis »_layout« deployed werden muss.

Sandboxed Solution dürfen aber keine Deployments in Verzeichnisse durchführen. Wenn Sie also Sandboxed Solution entwickeln wollen, dann müssen Sie mit einem leeren Projekt beginnen.

2. Bestätigen Sie Ihre Angaben mit OK. Sie werden nach dem Typ ihrer Solution gefragt. Die Abbildg. 26.41Abbildung 26.41 zeigt das entsprechende Dialogfeld.

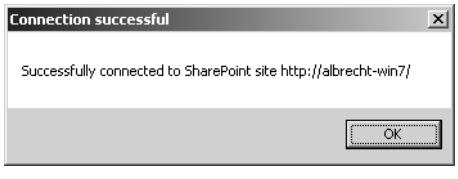
Abbildg. 26.41 Leeres SharePoint-Projekt – Standardeinstellungen



Sie sehen, dass die Sandbox Solution die Standardeinstellung für ein leeres Projekt ist.

3. Klicken Sie nun zur Validierung des Pfades zu Ihrer Entwicklungssite auf den auf die Schaltfläche Validate. Visual Studio wird diesen Pfad für das Deployment und auch das Debuggen der Solution verwenden.

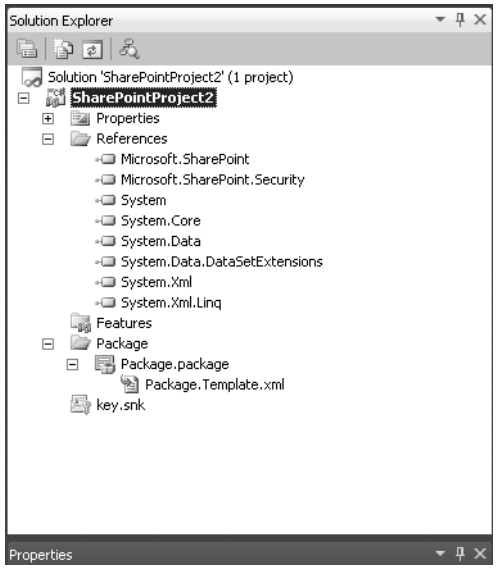
Abbildg. 26.42 Leeres SharePoint-Projekt – Erfolgreiche Validierung des Bereitstellungspfads



4. Bestätigen Sie das sich öffnende Dialogfeld (vgl. Abbildg. 26.42Abbildung 26.42) mit OK und klicken Sie abschließend auf Finish.

Anschließend wird das noch leere Projekt angezeigt. Interessant sind zunächst der Solution Explorer und das Properties-Fenster.

Abbildg. 26.43 Leeres SharePoint-Projekt – Solution Explorer



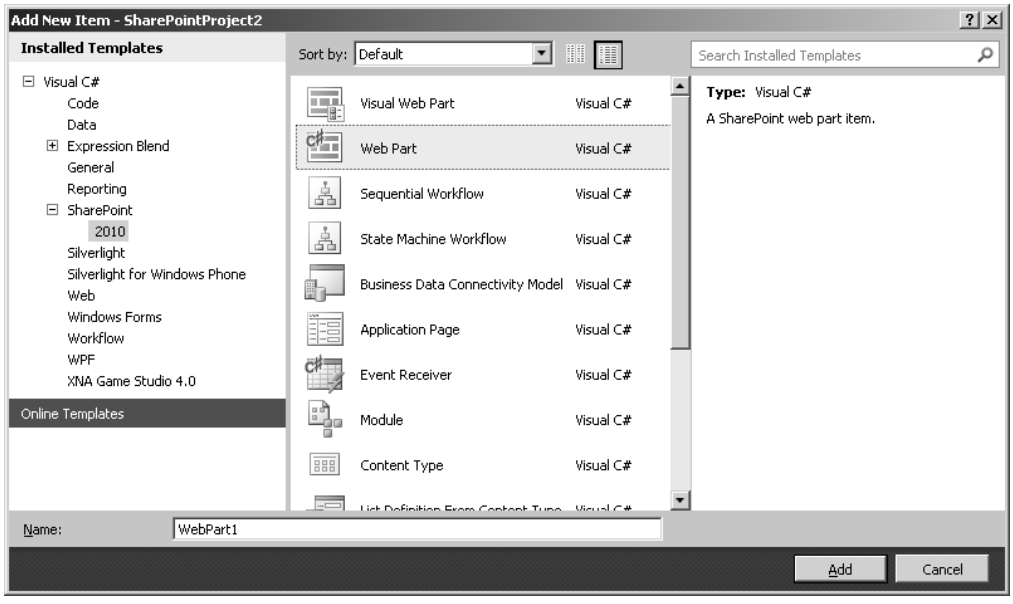
Sie sehen hier die Standardreferenzen eines .NET-Projekts und die Referenzen auf das SharePoint-Objektmodell und die SharePoint Security. Der noch leere Ordner Features wird später alle Features der Solution beinhalten. Im Package »Verzeichnis« werden alle Package-Information abgelegt. Zum Schluss sehen Sie die Datei key.snk – die Standarddatei für einen »Strong-Name«.

HINWEIS Eine Assembly »erhält« einen Strong Name durch das Hinzufügen eines 1.024-Bit-Schlüssels. Der Schlüssel hat nicht die Aufgabe, den Quellcode zu schützen, wie man vielleicht denken könnte. Strong Names sind dazu da, mehrere Assemblys mit gleichem Namen voneinander zu unterscheiden.

Fügen Sie jetzt einen Webpart hinzu.

1. Dazu klicken Sie Strg + ⇧ + A.
2. Im sich öffnenden Dialogfeld Add New Item wählen Sie den Knoten *SharePoint/2010*.
3. Wählen Sie in der Liste den Eintrag Web Part. Übernehmen Sie den vorgeschlagenen Namen.

Abbildg. 26.44 Einem Projekt ein Webpart hinzufügen



4. Bestätigen Sie durch Klicken auf die Schaltfläche Add.

Visual Studio hat nun folgendes angelegt:

- Ein Feature namens *Feature1*
- Eine Datei *Elements.xml*
- Die Codedatei *WebPart1.cs*
- Die Datei *WebPart1.webpart*

Jetzt können Sie den unten gezeigten Quellcode in die Datei *WebPart1.cs* eingeben. Dieser Code unterscheidet sich nicht von einem Code, den Sie für eine Farm Solution eingeben würden.

5. Geben Sie den in Listing 26.17Listing 26.17 dargestellten Code ein.

Listing 26.17 Quellcode für einen Sandboxed 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 de.meinefirma.SandboxedWebPart.WebPart1
{
    [ToolboxItemAttribute(false)]
    public class WebPart1 : WebPart
    {
```

Listing 26.17 Quellcode für einen Sandboxed Webpart *(Fortsetzung)*

```
        protected override void CreateChildControls()
        {
            // CreateChildControls der Basisklasse aufrufen
            base.CreateChildControls();

            // Label erzeugen
            Label lblTitel = new Label();
            // Label beschriften
            lblTitel.Text = "Übersicht Site Listen";
             // Label der Steuerlementeliste hinzufügen
            Controls.Add(lblTitel);

            // Als Demo zeige ich nun die Verwendung eines WebControls

            // das Control entspricht einem <br />
            WebControl zeilenumbruch = new WebControl(HtmlTextWriterTag.Br);


            // Zeilenumbruch der Steuerlementeliste hinzufügen
            Controls.Add(zeilenumbruch);

            // jetzt brauchen wir noch etwas, um die Liste anzuzeigen

             // in einer Listbox kann man Zeilen anzeigen
            ListBox box = new ListBox();

            // ich definiere so viele Zeilen, wie unser aktuelles Web Listen hat
            box.Rows = SPContext.Current.Web.Lists.Count;

// jetzt in einer foreach-Schleife Name und Beschreibung aller Listen in die box einfügen

            foreach (SPList liste in SPContext.Current.Web.Lists)
            {
                box.Items.Add(new ListItem(liste.Title, liste.Description));

            }
            // auch die Box in die Steuerlementeliste einbinden
            Controls.Add(box);
            // Ich signalisiere, dass die Erzeugung der ChildControls abgeschlossen ist
            ChildControlsCreated = true;
        }
    }
}
```

Einiges an diesem Quellcode kennen Sie schon aus den vorangegangenen Beispielen. Durch die Kommentierung im Quellcode wird die Verwendung der neuen Objekte

- WebControl
- HtmlTextWriterTag.Br
- ListBox
- SPContext

beschrieben.

In diesem Beispiel wurde auf eine Definition des SPSite und des SPWeb-Objekts verzichtet. Im Quellcode repräsentiert das SPContext-Objekt die Websitesammlung. Über »Current.Web« wird auf die Website zugegriffen, in der die Sandboxed Solution zukünftig deployed sein soll.

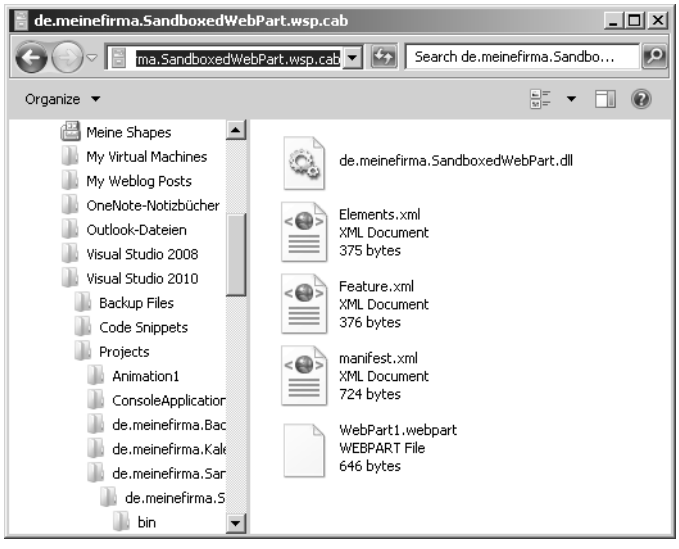
Das Ausrollen der Solution kann auf einfache Weise in Visual Studio erfolgen: Drücken Sie dazu lediglich die F5-Taste. Damit starten der Build-Prozess und anschließend das Deployment auf dem Entwicklungsserver. Gleichzeitig wechselt Visual Studio in den Debugger Modus.

Das Ergebnis des Builds ist ein Standard-SharePoint-Package mit der Erweiterung .wsp.

HINWEIS Die *.wsp*-Datei ist eine komprimierte Cabinet-Datei. Dateien dieses Typs haben normalerweise die Erweiterung *.cab* und können unter Windows durch Doppelklick geöffnet werden.

Sie können die *.wsp*-Datei in eine Cabinet-Datei umbenennen und dann den Inhalt durch Doppelklick öffnen. Das Ergebnis entspricht dann Abbildg. 26.45Abbildung 26.45.

Abbildg. 26.45 Eine entkomprimierte *.wsp*-Datei



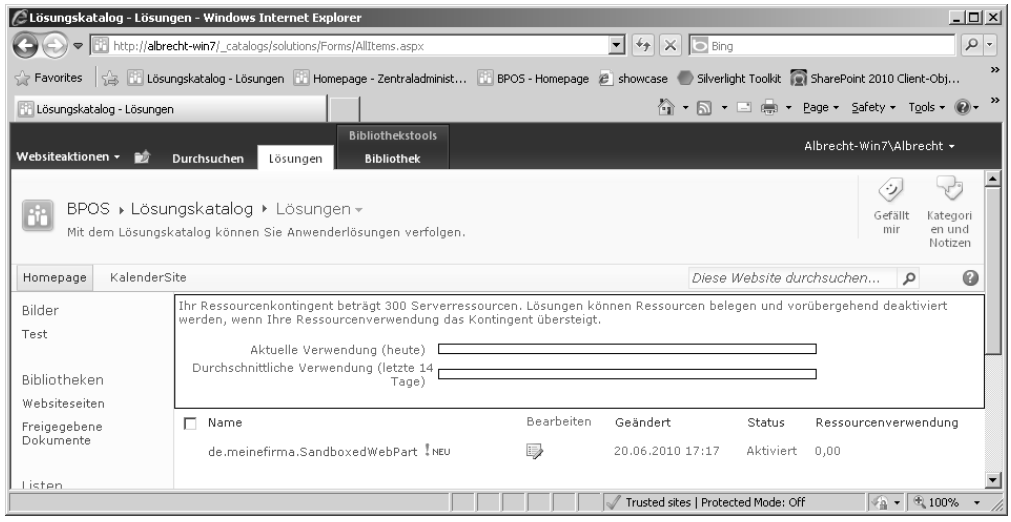
Visual Studio startet am Ende des Prozesses automatisch die Entwicklungs-Website im Browser.

HINWEIS Sie werden dabei eventuell aufgefordert, ihren Benutzernamen und das Passwort einzugeben.

Wechseln Sie in den SharePoint Lösungsbereich:

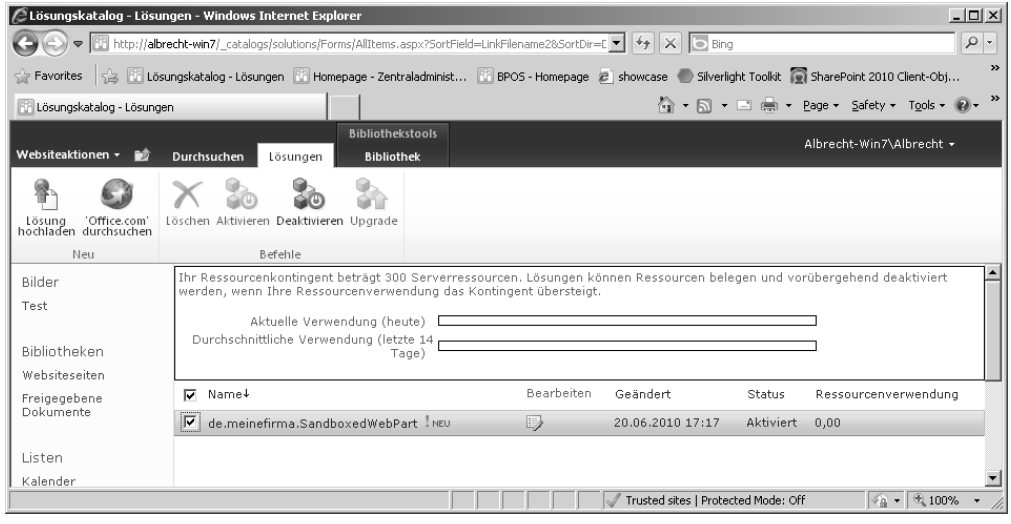
1. Öffnen Sie die SharePoint Seite und wählen Sie den Menübefehl *Websiteaktionen/Websiteeinstellungen*.
2. Öffnen Sie die Abbildg. 26.46Abbildung 26.46 dargestellte Galerienliste *Lösungen*. Sie sehen jetzt die neue Solution.

Abbildg. 26.46 Sandboxed Solution-Galerie öffnen



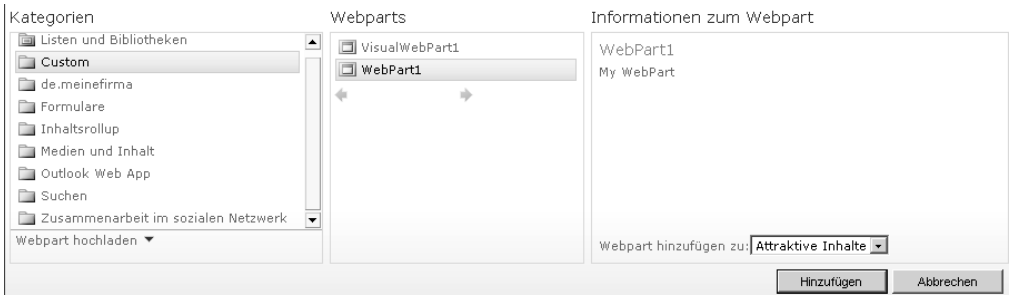
3. Markieren Sie die Solution. Abbildg. 26.47Abbildung 26.47 zeigt im Menüband Lösungen die aktivierte Schaltfläche Deaktivieren. Sie sehen, dass mit dem Upload eine Sandboxed Solution automatisch aktiviert ist.

Abbildg. 26.47 Sandboxed Solution zum Bearbeiten markieren



4. Wählen Sie jetzt *Websiteaktionen/Seite bearbeiten*. Klicken Sie anschließend *Einfügen/Webpart*. Im Ordner *Custom* finden Sie die entwickelte Sandboxed Solution »WebPart1«.

Abbildg. 26.48 Sandboxed Solution einfügen



5. Markieren Sie WebPart1, klicken Sie auf Hinzufügen und abschließend auf Speichern und schließen.

Abbildg. 26.49 Einfügen der Sandboxed Solution abschließen

Ab jetzt verbraucht das Webpart Ressourcen, zwar minimal, aber immerhin. Die Abbildg. 26.50Abbildung 26.50 zeigt den Stand unmittelbar nach dem Deployment.

Abbildg. 26.50 Ressourcenverbrauch einer Sandboxed Solution

6. Wechseln Sie wieder zu Visual Studio und beenden Sie mit *Debug/Stop Debugging* den Debugger.

Damit wird auch die Solution wieder aus der Solution-Galerie entfernt. Seien Sie also nicht überrascht, wenn Sie nun auf Ihrer Seite einen Fehler angezeigt bekommen. Die Fehlermeldung ist eindeutig. Die Solution kann nicht gefunden werden.

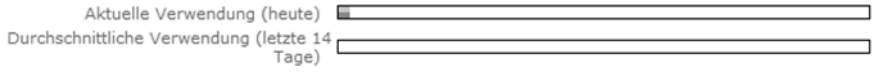
Um den Ressourcenverbrauch zu zeigen, wurde in diesem Beispiel eine Sandboxed Solution geschrieben, die 1000 Listen löscht (siehe Listing 26.18Abbildung 26.18). Die Listen heißen Kunden0 bis Kunden999 und wurden zuvor angelegt.

Listing 26.18 Webpart mit erhöhtem Ressourcenverbrauch

```
public class WebPart1 : WebPart
    {
        protected override void CreateChildControls()
        {

            SPContext.Current.Web.AllowUnsafeUpdates = true;
            // CreateChildControls der Basisklasse aufrufen
            base.CreateChildControls();
            TextBox text = new TextBox();
            int i = 0;
            int counter = 0;
            int max = SPContext.Current.Web.Lists.Count - 1;
            for (i = max; i > 0; i--)
            {
                SPList liste = SPContext.Current.Web.Lists[i];
                if (liste.Title.Contains("Kund"))
                {

                    counter++;
                    liste.Delete();
                    SPContext.Current.Web.Update();
                }
            }
            text.Text = "Es wurden " + counter + " Listen gelöscht";
            // auch die Box in die Steuerlementeliste einbinden
            Controls.Add(text);
            // Ich signalisiere, dass die Erzeugung der ChildControls abgeschlossen ist
            ChildControlsCreated = true;

        }
```

Nach der Installation der Solution zeigt die Solution Galerie einen Ressourcenverbrauch von 6,01 an.

Abbildg. 26.51 Erhöhter Ressourcenverbrauch einer Sandboxed Solution



Es ist also durchaus denkbar, dass eine Solution vom System deaktiviert werden kann. Deshalb müssen Sandboxed Solution so programmiert werden, dass sie möglichst wenige Ressourcen verbrauchen.

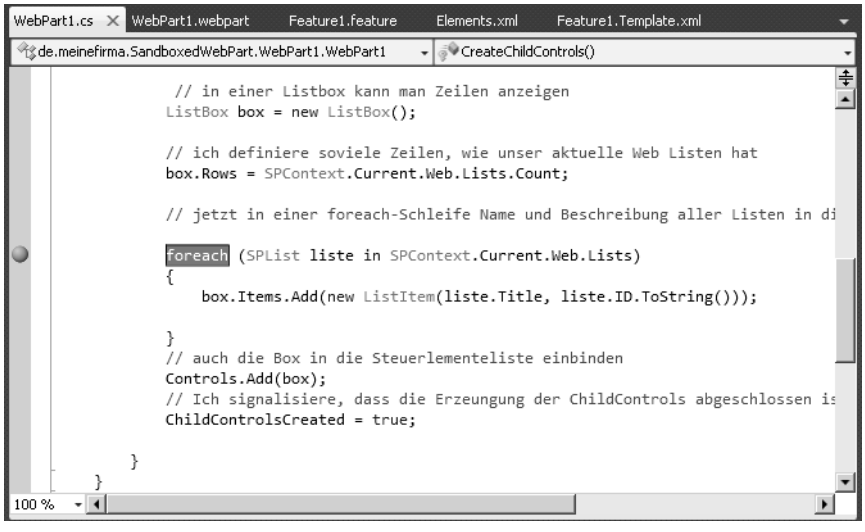
Debugging einer Sandboxed Solution

Da Programmcode immer Fehler enthält, ist das Debugging, die Fehlersuche, eine der Hauptbeschäftigungen eines Entwicklers. Nachfolgend zeigen wir Ihnen, wie Visual Studio die Fehlersuche unterstützt und werden dies am Beispiel unserer Sandboxed Solution demonstrieren.

Dazu wird die »foreach-Schleife« analysiert. In der linken Markierungsspalte des Editors wurde ein Breakpoint gesetzt. Wenn Sie das Beispiel nachvollziehen möchten, gehen Sie so vor:

1. Klicken Sie mit der Maus in diese Spalte – etwas in Höhe der foreach-Zeile. Die Ansicht sollte sich entsprechend Abbildg. 26.52Abbildung 26.52 darstellen.
2. Durch nochmaliges Klicken kann der Breakpoint wieder entfernt werden.

Abbildg. 26.52 Breakpoint setzen

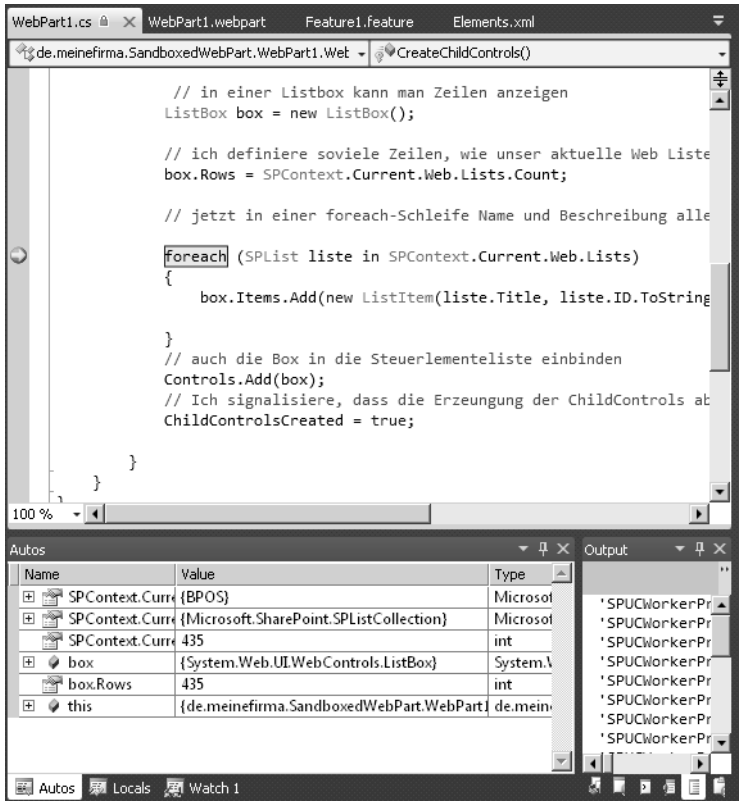


3. Drücken Sie jetzt die F5-Taste. Im Unterschied zu den ersten Tests wird zwar der Browser geöffnet, aber Sie sehen nur einen weißen Bildschirm. Das hängt mit dem Breakpoint zusammen.
4. Wechseln Sie zu Visual Studio.

Sie sehen, dass das Programm in der foreach-Zeile steht. Ein gelber Pfeil im Breakpoint signalisiert, dass die Anwendung an dieser Stelle gestoppt hat.

Unterhalb des Editorfensters sehen Sie die »View Autos«. Hier wird der aktuelle Inhalt der Objekte angezeigt. So können Sie erkennen, dass in der ListBox »box 435« Listen enthalten sind.

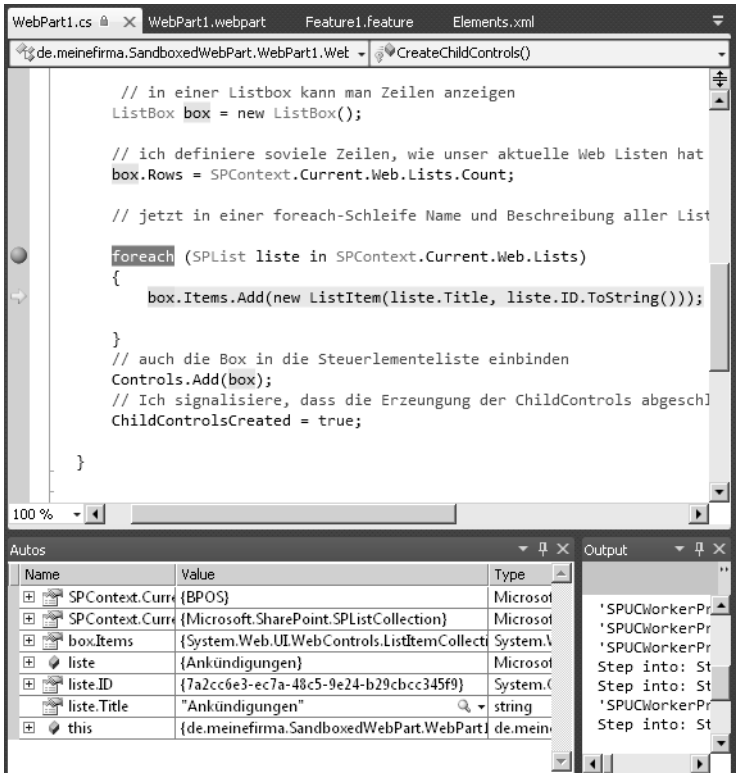
Abbildg. 26.53 Debugger stoppt am Breakpoint



5. Drücken Sie nun die Taste F11, um das Programm in Einzelschritten abzuarbeiten. Der Cursor wechselt in die Schleife, so dass Sie (vgl. Abbildg. 26.54Abbildung 26.54) im unteren linken Fenster den Namen und die aktuellen Werte aller beteiligten Objekte sehen können.

In Abbildg. 26.54Abbildung 26.54 der Entwicklungsumgebung können Sie sehen, dass die erste Liste die Liste »Ankündigungen« ist.

Abbildg. 26.54 Debugger in der ersten Schleife



Sie können sich vorstellen, welcher Vorteile der Debugger bietet, wenn die Zahl der Zeilen und der Variablen einer Solution ansteigt.

TIPP Wenn Sie komplexere Anwendungen schreiben oder bestehende Anwendungen erweitern möchten, dann müssen Sie sich mit dem Debugger vertraut machen.

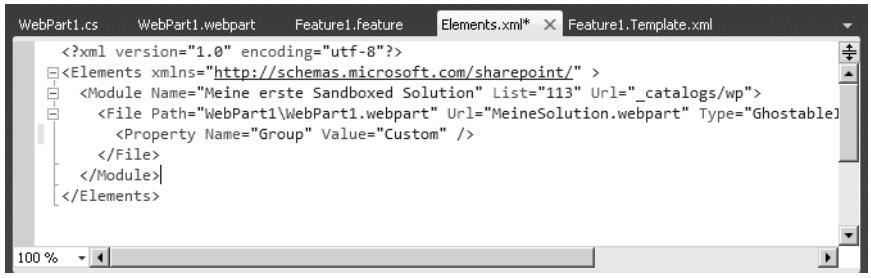
Konfiguration einer Sandboxed Solution

Zum Abschluss möchten wir noch kurz zeigen, wie Sie die Sandboxed Solution konfigurieren können. Bisher haben wir unsere Lösungen immer in einer Gruppe gefunden, die »Custom« heißt.

In größeren Projekten wäre es durchaus wünschenswert, das Solutions nach Kriterien wie z.B. dem »namespace« gruppiert werden, um Sie leichter zu verwalten. Die Vorgehensweise ist hier folgende:

1. Öffnen Sie im Solution Explorer des Visual Studio Projekts den Knoten WebPart1. Hier finden Sie die Datei *Elements.xml*, deren Inhalt Sie in Abbildg. 26.55Abbildung 26.55 sehen.

Abbildg. 26.55 *Elements.xml* im Visual Studio Editor



2. Nehmen Sie hier für die Property Group folgende Änderung vor:

```
<Property Name="Group" Value="de.meinefirma" />
```

3. Speichern Sie anschließend und drücken Sie die Taste F5.

Sie finden jetzt in der Liste *Kategorien* den Ordner »de.meineFirma«.

Abbildg. 26.56 Konfigurierte Sandboxed Solution

Wenn Ihnen Standardname und -Beschreibung nicht passen, ändern Sie diese in der Datei WebPart1.webpart. Diese XML-Datei hat den folgenden Inhalt.

Listing 26.19 Inhalt der Datei *WebPart1.webpart*

```
<?xml version="1.0" encoding="utf-8"?>
<webParts>
  <webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
    <metaData>
      <type name="de.meinefirma.SandboxedWebPart.WebPart1.WebPart1,
$SharePoint.Project.AssemblyFullName$" />
      <importErrorMessage>$Resources:core,ImportErrorMessage;</importErrorMessage>
    </metaData>
    <data>
      <properties>
        <property name="Title" type="string">MeinWebPart</property>
        <property name="Description" type="string">einerstes Beispielt</property>
```

Listing 26.19 Inhalt der Datei *WebPart1.webpart* *(Fortsetzung)*

```
      </properties>
    </data>
  </webPart>
</webParts>
```

Die Änderung könnte so aussehen.

Listing 26.20 Geänderte Datei *WebPart1.webpart*

```
<properties>
        <property name="Title" type="string">MeinWebPart</property>
        <property name="Description" type="string">einerstes Beispielt</property>
      </properties>
```

Bewertung

Der große Vorteil einer Sandboxed Solution besteht in der schnelleren Entwicklung von Business-Anwendungen. Schnell, weil das Deployment nicht mehr von der Farm Administration abhängt. Da Sandboxed Solutions nahezu alle Anwendungsfälle abdecken, deren Anwendungsgebiet in einer Websitesammlung liegt, gilt es als Best Practice, immer mit einer Sandboxed Solution zu beginnen.

Silverlight

Silverlight ist seit dem ersten Release im Jahr 2007 eine schnell wachsende Technologie. Ursprünglich bestand die Intention von Microsoft bei Silverlight darin, Anwendungen mit Multimediaelementen und Animationen anzureichern. Zielsystem war das Internet. Seit Silverlight 3 zielt die Technologie auf Businessanwendungen und ist damit insbesondere für SharePoint von größtem Interesse. Visual Studio und Microsoft Expression Blend sind hier äußerst leistungsfähige Werkzeuge.

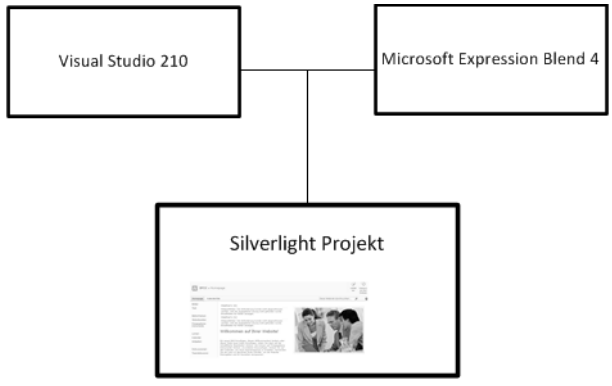
In diesem Kapitel werden wir an Hand einfacher Beispiele die Programmierung von Silverlight unter SharePoint 2010 demonstrieren. Ein vollständiger Überblick würde den Umfang des vorliegenden Buches sprengen.

Für die Silverlight-Entwicklung erweitern wir das Programmiermodell um die Anwendung »Microsoft Expression Blend«.

HINWEIS Mit Expression Blend erstellen Sie die grafische Benutzeroberfläche. Da Microsoft Expression Blend und Visual Studio die Arbeit am gleichen Projekt ermöglichen, können Sie zwischen beiden Programmen in der Entwicklungsphase wechseln. Ein weiterer Vorteil besteht darin, dass Designer und Entwickler jetzt problemlos am gleichen Projekt arbeiten können.

Um diese Zusammenarbeit zu demonstrieren, werden wir in den beiden Silverlight-Projekten die Oberfläche mit Expression Blend gestalten und die Logik dann in Visual Studio implementieren.

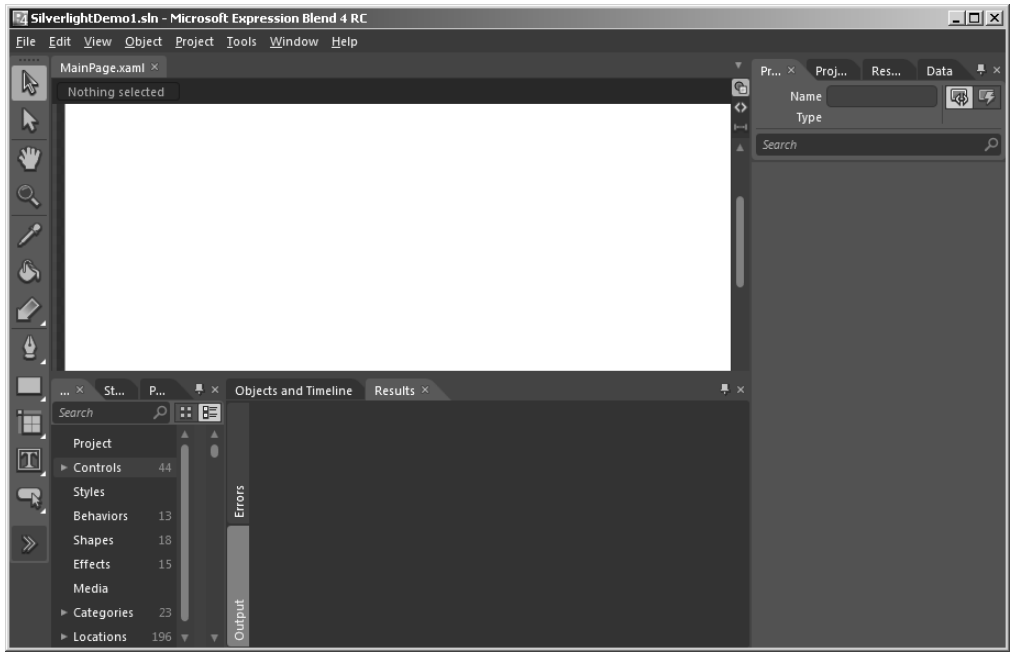
Abbildg. 26.57 Silverlight-Entwicklung mit Expression Expression Blend 4 und Visual Studio 2010



HINWEIS Für Microsoft Expression Blend spricht auch die Tatsache, dass selbst anspruchsvolle Anwendungen fast ohne das Schreiben von Codezeilen umgesetzt werden können.

Die folgende Abbildg. 26.58Abbildung 26.58 zeigt die Oberfläche von Expression Blend nach dem Anlegen eines neuen Projekts.

Abbildg. 26.58 Ein Silverlight-Projekt in Expression Blend



Im Folgenden beschreiben wir zwei Projekte. Ein *Taschenrechner* zeigt die Programmierung von Oberflächen und das Projekt *ListBox* zeigt den Einsatz von Web Services.

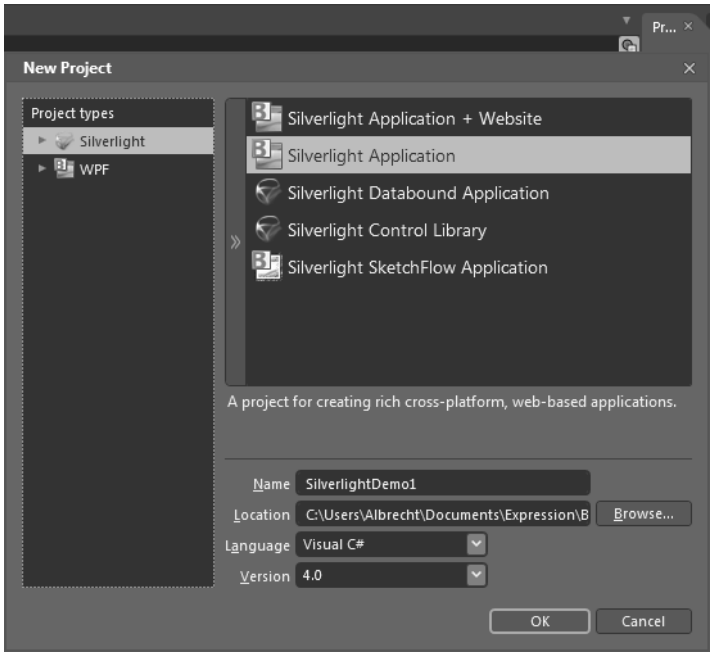
Ein erstes Projekt – ein kleiner Taschenrechner

In diesem Kapitel erfahren Sie, wie Sie einen kleinen Taschenrechner programmieren können. Sie sehen, wie die Steuerelemente eingefügt werden und wie die Interaktion zwischen den Steuerelementen programmiert werden kann.

So gehen Sie vor:

1. Starten Sie Expression Blend und legen Sie über den Menübefehl *File/New Project* ein neues Silverlight-Projekt an.
2. Wählen Sie als Project Typ *Silverlight* und als Vorlage *Silverlight Application*.
3. Schließen Sie den Vorgang mit OK ab.

Abbildg. 26.59 Ein *Silverlight Application*-Projekt



Als erstes muss nun die Oberfläche gezeichnet werden. Dazu benötigen Sie »TextBoxes«, »Labels und »Buttons«.

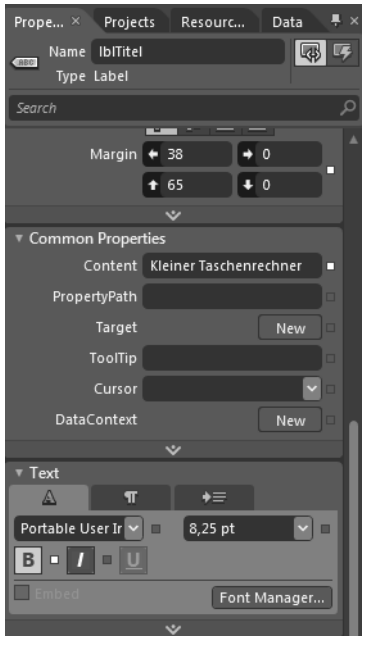
4. Klicken Sie auf A*ssets/Controls,* um die Controls zu öffnen.
5. Öffnen Sie den Knoten *Controls* und ziehen Sie nacheinander drei TextBoxes, vier Labels und zwei Buttons auf die weiße Zeichenfläche.
6. Ordnen Sie die Elemente entsprechend Abbildg. 26.60Abbildung 26.60 an.

Abbildg. 26.60 Layout des Taschenrechners

Um die Labels zu beschriften, gehen Sie so vor:

1. Markieren Sie per Selektions-Werkzeug (linker Rand) das erste Label.
2. Nehmen Sie über die rechte Spalte, im Register Properties, Formatierungen für die Eigenschaften wie Name, Inhalt (Content) und Formate vor. (vgl. Abbildg. 26.61Abbildung 26.61)

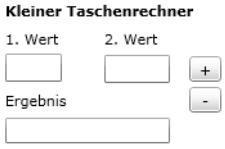
Abbildg. 26.61 Titel des Taschenrechners formatieren



3. Wiederholen Sie den Vorgang für die anderen Labels.
4. Löschen Sie zusätzlich den Inhalt der TextBoxes.

Das Ergebnis könnte sich entsprechend Abbildg. 26.62Abbildung 26.62 darstellen.

Abbildg. 26.62 Formatierter Taschenrechner



5. Mit F5 können Sie die Anwendung für einen ersten Test im Browser starten.

HINWEIS Nehmen Sie sich Zeit und experimentieren Sie mit den Möglichkeiten, die das Properties-Fenster für die Formatierung von Objekten bietet. Setzen Sie Eigenschaften wie Hintergrundfarbe, Schriftfarbe und Schriftgröße.

Mit der Formatierung der Steuerelemente ist das Design des Taschenrechners abgeschlossen.

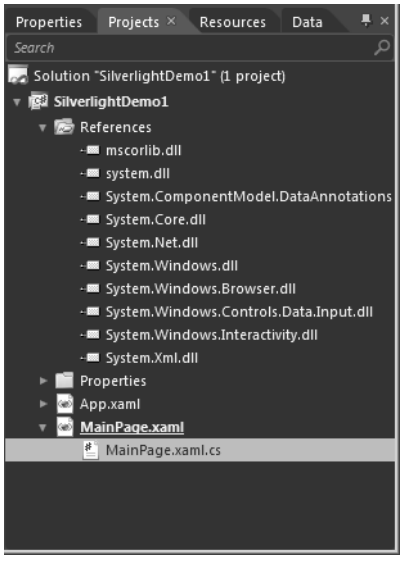
Struktur der Silverlight Solution

Um die Struktur einer Silverlight Solution einzusehen, gehen Sie so vor:

1. Klicken Sie in der rechten Spalte auf die Registerkarte *Projects*, um den Solution Explorer einzublenden.

2. Öffnen Sie den Ordner *References* und den Knoten MainPage.xaml.cs, wie in Abbildg. 26.63Abbildung 26.63 dargestellt.

Abbildg. 26.63 Inhalt des Solution Explorers für das Projekt *SilverlightDemo1*



In den References finden Sie keinen Hinweis auf SharePoint, weil Silverlight eine Clienttechnologie ist. Für unser Thema bedeutet dies, dass Silverlight als Multimediakomponente in eine ASPX-Seite eingebunden wird. Eine Verbindung zu SharePoint wird dann hergestellt, wenn das Silverlight Modul via Web Services Dienste von SharePoint nutzt, um zum Beispiel die Listen einer Site anzuzeigen. Für den Taschenrechner gibt es keinen Grund, SharePoint Services in Anspruch zu nehmen. Unser Taschenrechner wird lediglich eine Komponente auf einer ASPX-Seite sein, die in SharePoint gehostet wird.

Neu ist für Sie die Datei *MainPage.xaml*. Zu dieser Datei gehört eine C#-Datei *mainpage.xaml.cs*. Dieses Prinzip kennen Sie schon. In der C#-Datei ist der Code Behind für die XAML-Datei abgelegt. In Listing Listing 26.2126.21 sehen Sie den Inhalt der XAML-Datei.

Listing 26.21 Quellcode der XAML-Datei

```
<UserControl
         xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
         xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
         xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008" xmlns:mc="http://
schemas.openxmlformats.org/markup-compatibility/2006" mc:Ignorable="d"
         x:Class="SilverlightDemo1.MainPage"
         Width="640" Height="480">

         <Grid x:Name="LayoutRoot" Background="White">
                  <TextBox x:Name="txtWert1" HorizontalAlignment="Left" Height="22"
Margin="38,106,0,0"
                 TextWrapping="Wrap" VerticalAlignment="Top" Width="44"/>
                  <TextBox x:Name="txtWert2" HorizontalAlignment="Left" Height="22"
Margin="116,107,0,0"
                 TextWrapping="Wrap" VerticalAlignment="Top" Width="51"/>
                  <TextBox x:Name="txtErgebnis" HorizontalAlignment="Left" Height="20"
Margin="38,156,0,0"
                 TextWrapping="Wrap" VerticalAlignment="Top" Width="128"/>
                <Button x:Name="BtnSub" Content="-" HorizontalAlignment="Left" Height="20"
               Margin="181,0,0,328" VerticalAlignment="Bottom" Width="25"
               />
               <Button x:Name="btnAdd" Content="+" HorizontalAlignment="Left" Height="20"
               Margin="181,108,0,0" VerticalAlignment="Top" Width="25"
               />
                 <sdk:Label x:Name="lblTitel" HorizontalAlignment="Left"
                  Margin="38,65,0,0" VerticalAlignment="Top" Width="168" Content="Kleiner
Taschenrechner" FontWeight="Bold"/>
                 <sdk:Label HorizontalAlignment="Left" Height="20" Margin="38,135,0,0"
                  VerticalAlignment="Top" Width="111" Content="Ergebnis"/>
                 <sdk:Label HorizontalAlignment="Left" Height="17" Margin="38,87,0,0"
                  VerticalAlignment="Top" Width="74" Content="1. Wert"/>
                 <sdk:Label HorizontalAlignment="Left" Height="17" Margin="115,87,0,0"
                  VerticalAlignment="Top" Width="69" Content="2. Wert "/>
         </Grid>
</UserControl>
```

Das nachfolgende Listing Listing 26.2226.22 zeigt den zugehörigen C#-Quellcode.

Listing 26.22 Quellcode zu *MainPage.xaml*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;

namespace SilverlightDemo1
{
 public partial class MainPage : UserControl
 {
   public MainPage(){
   // Required to initialize variables
      InitializeComponent();
 }
}
```

Einige Erläuterungen hierzu:

- Sie sehen im Quellcode, dass im Konstruktor der Klasse »MainPage« die Methode »InitializeComponent()« aufgerufen wird. Diese initialisiert alle Komponenten des Moduls.
- Die Komponenten selbst sind in der XAML-Datei definiert. Am Quellcode in Listing 26.21Listing 26.21 erkennen Sie, dass es sich um eine XML-Datei handelt. Die Datei definiert ein »UserControl«, das aus einer Tabelle besteht, die die Steuerelemente unseres Taschenrechners beinhaltet.
- Die XAML-Datei definiert also deklarativ das Aussehen der Oberfläche. Die Oberfläche kann damit ganz ohne Programmierlogik definiert werden.

Exkurs XAML

Extensible Application Markup Language, XAML, ist eine deklarative Sprache, mit deren Hilfe Steuerelemente wie Schaltflächen, Listenfelder, Tabelle und vieles mehr definiert werden können. Der Vorteil von XMAL besteht darin, dass die Definition der GUI, das Aussehen der Anwendung, von der Logik der Software getrennt werden kann. Die Beschreibung der Oberfläche wird in einer XAML-Datei gespeichert. Dazu gibt es eine »Code-Behind-Datei«, welche die Logik der Interaktion zwischen den Komponenten definiert.

Bei der Anwendungsentwicklung ist XAML die gemeinsame Sprache zwischen Designer und Entwickler. Der Grafiker gestaltet mit seinen Designertools, in unserem Zusammenhang ist das Expression Blend, die Benutzerschnittstelle und generiert XAML-Code. Der Entwickler greift diesen XAML-Code auf und entwickelt in seiner Sprache der Wahl die Programmlogik dazu. XAML-Objekte, die vom Designer angelegt wurden, werden vom Entwickler mit programmlogischer Funktionalität versehen. Beide, Designer und Entwickler, können mit Expression Blend und Visual Studio 2010 gleichzeitig am selben Projekt arbeiten und müssen nicht mehr über Umwege miteinander kommunizieren. ▶

XAML-Dateien sind hierarchisch strukturiert. Ein oder mehrere Elemente können, abhängig von ihrer Ordnung, das Layout und Verhalten der Oberfläche beeinflussen. Jedes Element besitzt nur ein Elternelement. Jedes Element kann eine unbegrenzte Anzahl weiterer Elemente besitzen. Nur bei einigen wenigen ist die Anzahl eingeschränkt, z.B. besitzt eine Bildlaufleiste (Scrollbar) kein weiteres Element. In allen XAML-Anwendungen ist das Wurzelobjekt typischerweise ein Panel oder ein von Panel abgeleitetes Objekt, das sich um Positionierung und Rendern der Inhalte kümmert. Ein Panel kann wiederum mehrere Panels beinhalten.

Eigenschaften und Einstellungen der Steuerelemente werden im Tag als Attribute aufgeführt. Wie jede XML-Datei besitzt XAML ein Root-Element. In XAML-Dokumenten muss das Root-Tag die Attribute »xmlns« und xmlns:x« besitzen, die dem Parser Informationen über die zu verwendenden Namensräume liefern.

In Silverlight Anwendungen ist das Root-Element immer das Element <UserControl>.

Die Silverlight XAML-Datei ist damit immer eine Erweiterung des in Listing 26.23Listing 26.23 gezeigten Quellcodes.

Listing 26.23 Silverlight-XAML

```
<UserControl x:Class="listBox.Page"
 xmlns="http://schemas.microsoft.com/client/2007"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
     <Grid x:Name="LayoutRoot" Background="White">

     </Grid>
</UserControl>
```

Der vorliegende Quellcode definiert, dass eine Klasse angelegt werden muss, die die im Grid-Element definierten Steuerelemente beinhaltet.

Programmieren der Interaktion

Der Taschenrechner benötigt eine gewisse Logik, um seine Aufgaben zu erfüllen. Dazu müssen zwei Dinge programmiert werden:

- Die Fähigkeit der Schaltflächen, auf Mausklick zu reagieren.
- Je nachdem, welche Schaltfläche geklickt wird, muss der Inhalt der beiden Textfelder dann berechnet und in das Ergebnis geschrieben werden.

1. Erweitern Sie deshalb zunächst den XAML-Quellcode wie in Listing 26.24Listing 26.24 dargestellt.

Die beiden XAML-Elemente für die Definition eines Buttons werden dabei jeweils um das Attribut »Click« erweitert. Der Wert des Attributs ist der Name einer Methode, die immer dann aufgerufen wird, wenn der Anwender eine der Schaltflächen klickt.

Listing 26.24 Event Handler hinzu fügen

```
<Button x:Name="BtnSub" Content="-" HorizontalAlignment="Left" Height="20"
  Margin="181,132,0,0" VerticalAlignment="Top" Width="25"
  Click="BtnSub_Click"
   />
<Button x:Name="btnAdd" Content="+" HorizontalAlignment="Left" Height="20"
  Margin="181,108,0,0" VerticalAlignment="Top" Width="25"
  Click="btnAdd_Click"/>
```

In Listing Listing 26.2526.25 sehen Sie dann die Veränderungen, die Expression Blend in der Code Behind-Datei vorgenommen hat. Die Using-Anweisungen sind im Listing nicht dargestellt.

Listing 26.25 Quellcode der Events im der Code Behind-Datei

```
namespace SilverlightDemo1
{
...........public partial class MainPage : UserControl
...........{
......................public MainPage()
......................{
.........................// Required to initialize variables
.........................InitializeComponent();
......................}

        private void btnAdd_Click(object sender, RoutedEventArgs e)
        {
        }
        private void BtnSub_Click(object sender, RoutedEventArgs e)
        {
        }
...........}
}
```

2. Führen Sie im Solution Explorer einen Doppelklick auf die Code Behind Datei aus und ergänzen Sie die beiden Methoden wie in Listing 26.26Listing 26.26 gezeigt.

Listing 26.26 Events mit Funktionalität

```
private void btnAdd_Click(object sender, RoutedEventArgs e)
 {
   int ergebnis  = Int32.Parse(this.txtWert1.Text) + Int32.Parse(this.txtWert2.Text);
   this.txtErgebnis.Text = ergebnis.ToString();
```

Listing 26.26 Events mit Funktionalität

```
        }
private void BtnSub_Click(object sender, RoutedEventArgs e)
  {
   int ergebnis = Int32.Parse(this.txtWert1.Text) - Int32.Parse(this.txtWert2.Text);
   this.txtErgebnis.Text = ergebnis.ToString();
  }
```

Das Ergebnis ist ein funktionsfähiger Taschenrechner.

3. Für einen ersten Test drücken Sie die Taste F5.

Expression Blend startet einen lokalen Webserver und öffnet eine HTML-Seite, die die Silverlight-Anwendung beinhaltet.

Abbildg. 26.64 Silverlight-Taschenrechner im Test

Damit ist das Grundprinzip einer Silverlight-Anwendung beschrieben.

Silverlight, SharePoint und Web Services

Das nächste Projekt zeigt den Zusammenhang zwischen Silverlight und SharePoint. Auch hier können wir Ihnen nur einen ersten Einstieg vermitteln, der die Grundlage einer späteren Vertiefung darstellt.

Projektbeschreibung

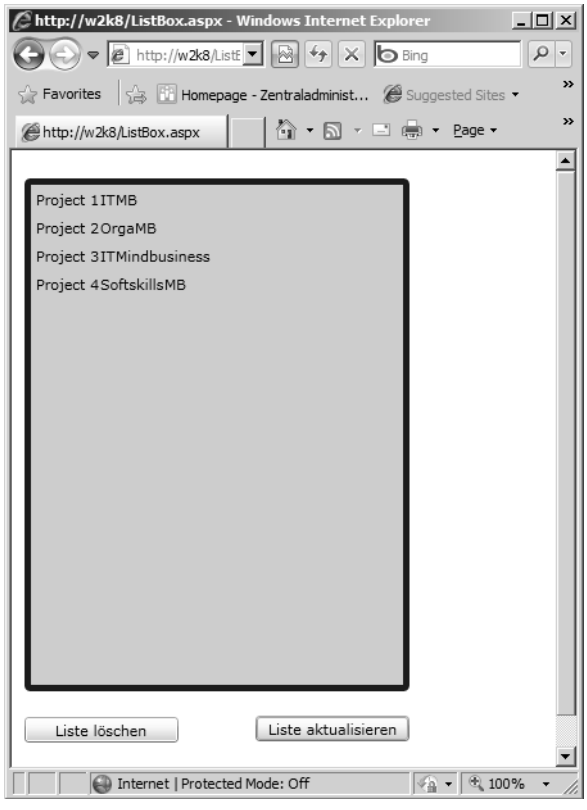
In diesem Projekt werden folgende Anwendungsfälle umsetzt:

- Die Spalten »Titel«, »Client« und »Type« der Liste »ProjectList« sollen in einem Fenster angezeigt werden.
- Beim ersten Laden der Seite soll die Liste schon vollständig ausgelesen sein.
- Mit Hilfe zweier Schaltflächen soll die Ausgabe der Liste gelöscht oder aktualisiert werden.

Bei der Umsetzung wird das Zusammenspiel zwischen Visual Studio und Expression Blend demonstriert. Das Beispiel ist relativ komplex zeigt aber den Zusammenhang zwischen Silverlight und SharePoint und vermittelt erste Programmierkenntnisse für das Arbeiten mit SharePoint Web Services.

Abbildung Abbildg. 26.6526.65 zeigt das Ergebnis der Anwendung, die in eine leere ASPX-Seite eingebunden ist.

Abbildg. 26.65 Die Silverlight Anwendung *ListBox*



Projektschritte

Um das Projekt zu realisieren, werden Expression Blend und Visual Studio eingesetzt. Es müssen dabei die folgenden Schritte umgesetzt werden:

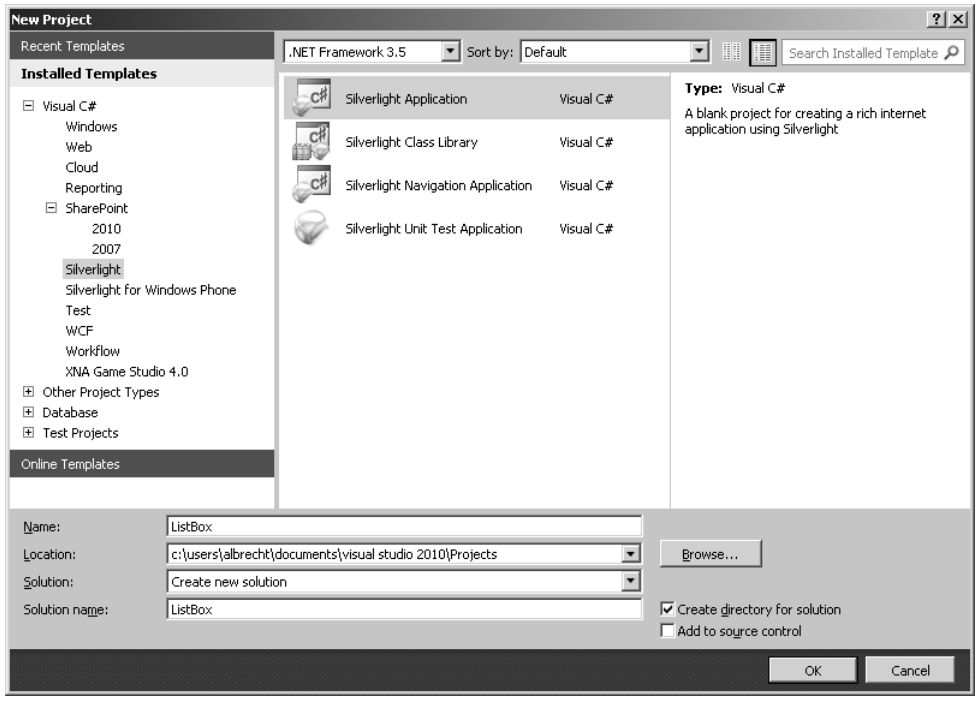
1. Erzeugen eines Silverlight Projekts
2. Definieren einer XAML-Datei für die GUI in Expression Blend
3. Einbinden des Lists Web Service in den Solution Explorer von Visual Studio
4. Schreiben des Quellcodes für die Code-Behind Datei in Visual Studio
5. Erzeugen einer ASPX-Seite
6. Import des Silverlight-Moduls in die SharePoint Website
7. Einbinden des Moduls in die SPX-Seite

Umsetzung des Projekts in Expression Blend und Visual Studio

So gehen Sie vor:

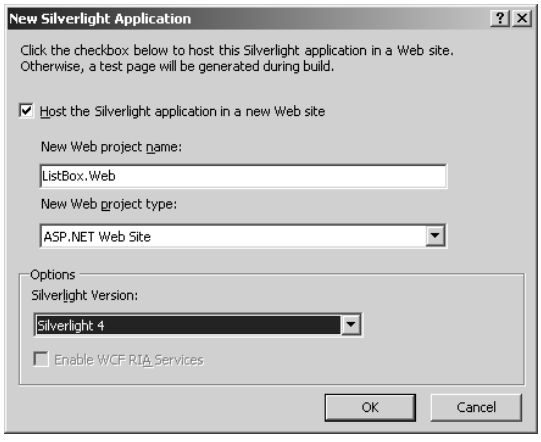
1. Starten Sie Visual Studio und legen Sie das Projekt »ListBox«, wie in Abbildg. 26.66Abbildung 26.66 dargestellt, an.

Abbildg. 26.66 Silverlight Projekt in Visual Studio anlegen



2. Klicken Sie auf OK und nehmen Sie die entsprechend Abbildg. 26.67Abbildung 26.67 dargestellten Einstellungen im Dialogfeld vor.

Abbildg. 26.67 Konfiguration der Silverlight-Anwendung



In Visual Studio sehen Sie jetzt das neue Projekt. Im Toolbox-Fenster werden Silverlight-Controls eingeblendet, die per Drag & Drop auf die Zeichenfläche gezogen werden können. Damit haben Sie auch in Visual Studio die Möglichkeit die XAML-Datei zu definieren. In unserem Beispiel werden wir dies jedoch mit Expression Blend umsetzen.

Definition der Benutzeroberfläche

Für die Definition der Benutzeroberfläche gehen Sie so vor:

1. Öffnen Sie im Solution Explorer das Kontextmenü der Datei MainPage.XAML.
2. Wählen Sie die Option *Open in Expression Blend*. Damit starten Sie das Projekt auch in Blend und können jetzt zwischen beiden Anwendungen wechseln.
3. Führen Sie in Expression Blend einen Rechtsklick in der Zeichenfläche aus und wählen Sie *View XAML*.
4. Ergänzen Sie das Grid-Element des generierten UserControl um die in Listing 26.27Listing 26.27 dargestellten Zeilen.

Listing 26.27 XAML Definition der Silverlight-Benutzeroberfläche

```
<UserControl x:Class="ListBox.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008" xmlns:mc="http://
schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="300" d:DesignHeight="500">
    <Grid x:Name="LayoutRoot" Background="White">
          <ListBox x:Name="_list" ItemsSource="{Binding Mode=OneWay}"
           Foreground="Black" BorderThickness="5" BorderBrush="Blue"
           Background="#FFCCCCD8" Height="400" Margin="0,0,0,100">
               <ListBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Horizontal">
                        <TextBlock Text="{Binding Title}"/>
                        <TextBlock Text="{Binding Type}"/>
                        <TextBlock Text="{Binding Client}"/>
```

Listing 26.27 XAML Definition der Silverlight-Benutzeroberfläche

```
                    </StackPanel>
                </DataTemplate>
            </ListBox.ItemTemplate>

        </ListBox>
        <Button Name="show" Content="Liste aktualisieren"  Click="show_Click" Height="20"
        Width="120" VerticalAlignment="Bottom" HorizontalAlignment="Right" ></Button>
        <Button Name="delete" Content="Liste löschen" Click="del_Click" Height="20"
        Width="120" VerticalAlignment="Bottom"  HorizontalAlignment="Left"></Button>

    </Grid>
</UserControl>
```

Die XAML-Datei lässt sich wie folgt beschreiben:

- Es werden die notwendigen Namespaces eingebunden
- Es wird ein UserControl ListBox als Klasse, Root-Element <UserControl> definiert
- Die Zeichenfläche in ein Gitternetz (Grid) mit der ID »LayoutRoot«
- Im Gitternetz wird eine ListBox gezeichnet, die die ID, »_list« besitzt.
- Der Inhalt der ListBox ist eine »ItemSource«, d.h. es werden Items einer Liste in die ListBox eingefügt. Die »ItemSource« ist die Datenquelle.
- Die Datenquelle wird unidirektional angebunden, also von der Quelle zur Liste. Änderungen in der »ListBox« werden demnach nicht in der Datenquelle, der SharePoint-Liste, aktualisiert.
- Innerhalb der Liste wird eine Vorlage für die Items der Liste, »ItemTemplate«, definiert. In der Vorlage sehen Sie als Kindelement eine Daten-Vorlage, »DataTemplate«. Das Item-Template definiert eine Vorlage für die Items der ListBox, das Daten-Template beschreibt, aus welchen Daten das Item besteht.
- Die Daten werden dann in einem sogenannten »Layout Container« angezeigt, dem StackPanel. Ein Layout Container ist für die Anordnung seiner Elemente zuständig. Wenn Sie sich weiter mit XAML auseinandersetzen, werden Sie feststellen, dass Sie mit Hilfe unterschiedlicher Layout Container sehr flexible Oberfläche bauen können, die sich auch dynamisch an verschiedene Fenstergrößen anpassen. In einem StackPanel werden die Inhalte als Zeilen horizontal oder vertikal angeordnet.
- Im Container werden drei Textblöcke definiert, die an die Spalten der SharePoint-Liste gebunden sind. Es wird also zuerst der Titel, dann der Typ und dann der Client in die Zeile geschrieben,
- Nach der Listbox folgt die Definition zweier Schaltflächen. Die Attribute definieren ID, Beschriftung, Höhe, Breite und weitere Eigenschaften. Für die Interaktion wichtig ist die Definition der Click-Attribute. Hier steht der Name eines Event-Handler. Der Event-Handler ist Programmcode, der ausgeführt werden soll, wenn die Schaltfläche geklickt wird.

HINWEIS In einem StackPanel kann der Inhalt vertikal oder horizontal angeordnet werden, Standard ist vertikal. Es würden also alle Listenelemente untereinander angeordnet werden. In unserem Beispiel wurde die horizontale Orientierung gewählt, um den Zeilencharakter der Listitems zu verdeutlichen.

Bevor wir uns nun den »Code-Behind« anschauen sollten wir uns nochmals vor Augen halten, dass XAML nur das Layout definiert, nicht die Logik. Der Quellcode hierzu wird in Visual Studio in der Datei MainPage.xaml.cs geschrieben.

Programmierung der Logik

Mit der Definition der XAML-Datei ist das Design abgeschlossen. Jetzt die Logik umgesetzt werden. Zu dieser Logik gehört der Remotezugriff auf SharePoint Server.

Für uns ist die Tatsache wichtig, dass SharePoint einen großen Teil seiner Features als Web Service zur Verfügung stellt. Das heißt, dass Sie prinzipiell auf alle Features von SharePoint zugreifen können, ohne dafür Anwendungen auf dem Server installieren zu müssen. Und das ist der springende Punkt. Wir haben keinen Zugriff auf das Dateisystem von SharePoint, wollen aber eigene Logik benutzen. Konsequenterweise setzen wir hierzu auf Web Services. Und Silverlight bietet die Möglichkeit, WebServices und .NET Framework zu nutzen.

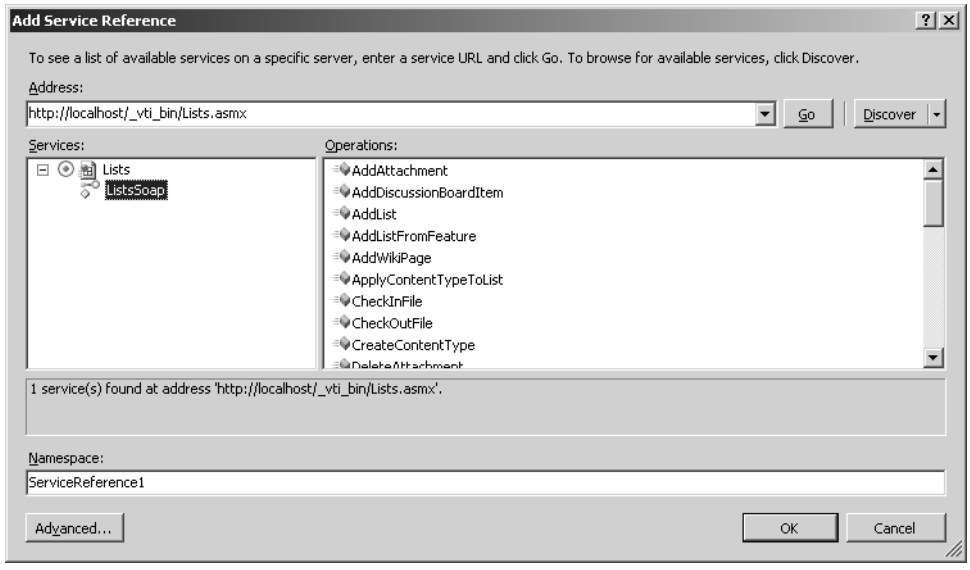
Im Code-Behind greifen wir auf den SharePoint Server zu und benutzen dazu einen SOAP-Client, der mit Hilfe eines Requests eine SharePoint-Liste als XML-File anfordert und verarbeitet. Dazu müssen entsprechende Bibliotheken eingebunden werden. In Visual Studio geschieht dies durch das Einbinden einer Service Reference.

So gehen Sie vor:

1. Nach dem Erstellen des Silverlight-Projekts öffnen Sie das Kontextmenü der Anwendung und wählen den Eintrag *Add Service Reference*.
2. Im sich öffnenden Dialogfeld müssen Sie die Adresse des Web Service angeben, den Sie nutzen wollen. In unserem Beispiel ist dass der »Lists-Service«.
3. Geben Sie in der Textbox *Address* die folgende URL ein: http://localhost/_vti_bin/Lists.asmx
4. Klicken Sie anschließend auf OK.
5. Öffnen Sie dann in der Spalte *Services* den Knoten *Lists* und klicken Sie auf *ListsSoap*.

Abbildung Abbildg. 26.6826.68 zeigt das Ergebnis. In der Spalte *Operations* werden alle Methoden des Service aufgeführt.

Abbildg. 26.68 Lists Services einbinden



Im Solution Explorer ist die Referenz auf den Web Service eingebunden. Jetzt können Sie auf die SharePoint-Listen so zugreifen, als liefe das Silverlight-Modul auf dem Server. Wie dies zu programmieren ist, sehen Sie in Listing 26.28Listing 26.28.

Listing 26.28 Zugriff auf SharePoint Web Services in einem Silverlight-Anwendung

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using System.Xml.Linq;
using ListBox.ServiceReference1;
namespace ListBox
{
public partial class MainPage : UserControl
    {
    ListsSoapClient proxy = new ListsSoapClient();
    XDocument doc = null;
    XElement query = null;
    XElement viewFields = null;
    XElement queryOptions = null;
    public MainPage()
    {
    InitializeComponent();
    proxy.GetListItemsCompleted +=
```

Listing 26.28 Zugriff auf SharePoint Web Services in einem Silverlight-Anwendung *(Fortsetzung)*

```
new System.EventHandler<GetListItemsCompletedEventArgs>(proxy_GetListItemsCompleted);
    doc = XDocument.Parse("<Document><Query /><ViewFields /><QueryOptions /></Document>");
   query = doc.Element("Query");
   viewFields = doc.Element("ViewFields");
   queryOptions = doc.Element("QueryOptions");
    proxy.GetListItemsAsync("ProjectList", "", query, viewFields, "10", queryOptions, "");
   }
 void proxy_GetListItemsCompleted(object sender, GetListItemsCompletedEventArgs e)
        {
           XElement result = e.Result;
           var Projects = from x in result.Elements().First().Elements()
                          select new ListResult
                           {
                               Title = x.Attribute("ows_Title").Value,
                               Client = x.Attribute("ows_Client").Value,
                               Type = x.Attribute("ows_Type").Value
                           };
            _list.DataContext = Projects;
        }
        public class ListResult
        {
            public string Title { get; set; }
            public string Client { get; set; }
            public string Type { get; set; }
        }

        private void show_Click(object sender, RoutedEventArgs e)
        {
            proxy.GetListItemsAsync("ProjectList", "", query, viewFields, "10",
queryOptions, "");
        }

        private void del_Click(object sender, RoutedEventArgs e)
        {
            _list.DataContext = "";
        }

    }
}
```

Neu sind die beiden Using-Direktiven:

```
using System.Xml.Linq;
using ListBox.ServiceReference1;
```

- Die erste Direktive bindet für das Parsen von XML-Dateien die LINQ-Bibliothek ein. LINQ, **L**anguage **IN**tegrated **Q**uery, ist eine Komponente von .NET Framework zur Abfrage von Datenquellen wie Datenbanken und XML-Dateien.
- Die zweite Direktive bindet einen Web Service ein, und zwar den SharePoint Web Service für die Abfrage und Bearbeitung von Listen.

HINWEIS Das Besondere dabei ist, dass die Kommunikation zwischen dem Browser und SharePoint folgende Merkmale hat:

- Der HTTP-Request ist ein »XMLHttpRequest«, d.h. der Datenaustausch erfolgt im »Hintergrund«. In der Adresszeile ändert sich nichts und nur Teile der Web-Seite werden neu aufgebaut
- Die Daten werden als XML-Daten übertragen. Die gilt für den Request und für den Response

Bei der Programmierung muss immer berücksichtigt werden, dass die Kommunikation asynchron erfolgt.

Die folgenden Ausführungen wenden sich an Entwickler. Für Nicht-Entwickler ist vor allem wichtig zu erkennen, dass im Quellcode XML-Daten verarbeitet werden. Die Verarbeitung erfolgt durch den Browser bzw. das Silverlight-PlugIn.

Der Quellcode kann wie folgt beschrieben werden.

Als erstes wird eine List SOAPClient »proxy« erzeugt, der die Kommunikation und die XML-Verarbeitung für uns übernimmt.

```
ListsSoapClient proxy = new ListsSoapClient();
```

Wir definieren ein XML-Dokument, eine XML-Abfrage, einen Container für Listfelder und ein XML-Element für Abfrage-Optionen.

Beim Erzeugen des Silverlight-Objekts geschieht nun Folgendes:

- Es wird ein EventHandler erzeugt, der immer dann aufgerufen wird, nachdem die SharePoint-Liste geladen wurde.

```
proxy.GetListItemsCompleted +=
new System.EventHandler<GetListItemsCompletedEventArgs>(proxy_GetListItemsCompleted);
```

- Dann wird im Konstruktor ein XML-Dokument erzeugt.
- Anschließend eine XML-Query.
- In gleicher Weise werden ViewFields und QueryOptions erzeugt. In ViewFields wird definiert, welche Spalten abgefragt werden soll, in den QueryOptions Optionen.
- Zum Schluss erfolgt ein asynchroner Request, der die Liste lädt und dabei den Web Service des SharePoint Servers nutzt.

```
proxy.GetListItemsAsync("ProjectList", „", query, viewFields, „10?, queryOptions, „");
```

Der erste Parameter übergibt den Namen der abzufragenden Liste. Diese Methode ruft automatisch nach dem Erhalt der Liste den EvenHandler

```
void proxy_GetListItemsCompleted(object sender, GetListItemsCompletedEventArgs e)
```

auf.

Dieser Event Handler wertet die übergebene Liste aus und schreibt sie in das Objekt Projekts, welches dann als Content der ListBox »_list« zugewiesen wird.

```
_list.DataContext = Projects;
```

Sie erinnern sich, »_list« ist die ListBox, die mit XAML in der Datei MainPage.xaml beschrieben wurde.

HINWEIS Der Ablauf ist knifflig, aber hoffentlich deutlich geworden. Einsteiger müssen sich natürlich noch intensiv mit der Syntax und den hier beschriebenen Objekten auseinander setzen.

Aus der Beschreibung des Quellcodes geht hervor, dass die Liste sofort dargestellt wird. Mit Hilfe der beiden Schaltflächen kann die Liste aktualisiert werden, indem die Proxy-Methode nochmals aufgerufen wird.

```
private void show_Click(object sender, RoutedEventArgs e)
{
proxy.GetListItemsAsync("ProjectList", "", query, viewFields, "10", queryOptions, "");
}
```

Der Quellcode der zweiten Schaltfläche setzt die ListBox zurück, indem ein leerer String zugewiesen wird.

```
private void del_Click(object sender, RoutedEventArgs e)
 {
 _list.DataContext = "";
}
```

Die Verarbeitung der Response ist in den folgenden Zeilen programmiert.

```
    XElement result = e.Result;
    var Projects = from x in result.Elements().First().Elements()
                select new ListResult
                {
                    Title = x.Attribute("ows_Title").Value,
                    Client = x.Attribute("ows_Client").Value,
```

```
                Type = x.Attribute("ows_Type").Value
            };
        _list.DataContext = Projects;
        }
```

Das Ergebnis der Response ist eine XML-Datei, die einen Knoten »Result« besitzt. Dieses Element wird dem XML-Element »result« zugewiesen, das dann in einem LINQ-Ausdruck verarbeitet wird. Dieser Ausdruck erzeugt eine Liste von ListResult-Objekten, die dann als Datenkontext der Liste »_list« zugewiesen werden.

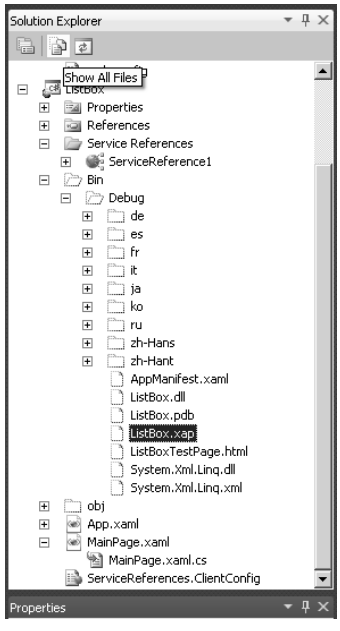
Die Klasse ListResult entspricht dem Aufbau der SharePoint-Liste »ProjectList«.

```
        public class ListResult
        {
            public string Title { get; set; }
            public string Client { get; set; }
            public string Type { get; set; }
        }
```

Jetzt kann die Anwendung kompiliert werden.

- Am schnellsten geschieht dies durch Drücken der Tastenkombination Strg+⇧+B.
- Blenden Sie jetzt im Solution Explorer alle Dateien ein und öffnen Sie den Ordner *Bin/Debug*. Hier finden Sie die Datei »ListBox.xap«. (siehe Abbildg. 26.69Abbildung 26.69)

Abbildg. 26.69 Silverlight-Anwendung nach dem Kompilieren



Diese Datei ist eine .zip-Datei, die alle Assemblys, Multimediadateien und Konfigurationsdateien beinhaltet, die auf dem Silverlight Client für die Programmausführung benötigt werden. Damit entspricht die XAP-Datei der WSP-Datei einer Sandboxed Solution. Allerdings wird diese Datei nicht bereitgestellt, sondern in eine *.aspx*-Datei eingebunden.

Die ASPX-Seite wird mit dem Share Point Designer geschrieben und in die Content-Datenbank von SharePoint importiert. Dabei schlüpfe man in die Rolle des Share Point Designers.

HINWEIS Diese Rolle wird in der Praxis von einer anderen Person übernommen werden, so dass wir hier nur die wichtigsten Schritte für den Überblick beschreiben. Je mehr Sie in die Programmierung des SharePoint 2010 einsteigen, desto mehr werden Sie auch in diese Rolle ausüben.

Silverlight-Anwendung veröffentlichen

In einem ersten Schritt wird eine .aspx-Seite in SharePoint Designer angelegt. Dazu wird SharePoint Designer gestartet und mit SharePoint verbunden. Über *Datei/Neu/Seite* wird die im Listing 26.29Listing 26.29 gezeigte *.aspx*-Seite mit dem Namen »testBox.aspx« definiert.

Listing Listing 26.2926.29 zeigt den hierfür benötigten Quellcode.

Listing 26.29 Ein Silverlight Modul in eine .aspx-Seite einbinden

```
<%@ Page Language="C#" %>
<html dir="ltr">
<head runat="server">
<META name="WebPartPageExpansion" content="full">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>Eingabemaske Neues Mitglied im DSD</title>
</head>
<body style="margin: 0; background-color: #E7E3E7">

<object data="data:application/x-silverlight," type="application/x-silverlight-2"
width="300" height="500">
<param name="source" value="ListBox.xap"/>
<param name="onerror" value="onSilverlightError" />
<param name="background" value="#E7E3E7" />
</object>
</body>
</html>
```

Für Webentwickler ist das Prinzip nicht neu.

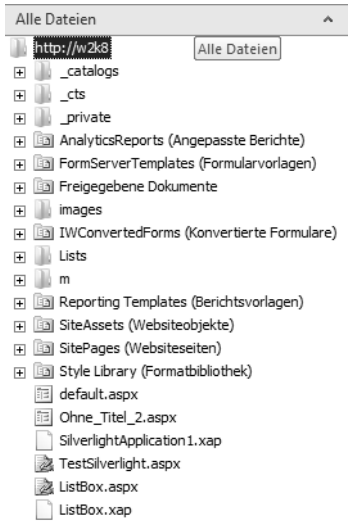
- In einem Object-Tag wird eine Quelle referenziert, in diesem Falle die »ListBox.xap«.
- Über »width« und »height« wird die Größe der Zeichenfläche definiert.
- Deren Inhalt und Interaktion wird vom Silverlight-Modul bestimmt.
- Das Modul kann nun permanent erweitert werden, die *.aspx*-Seite bleibt immer gleich.

Als nächstes wird die im Verzeichnis *Bin/Debug* liegende XAP-Datei über *Datei/Seiten/Dateien importieren* importiert.

Das Ergebnis sehen Sie in Abbildg. 26.70Abbildung 26.70. Die *.aspx*-Datei und die XAP-Datei liegen im Root der SharePoint-Site.

TIPP In der Praxis empfehlen wir, immer ein Verzeichnis anzulegen, in dem alle Silverlight-Module abgelegt werden können. So kann auch über Verzeichnisse die Architektur einer Site abgebildet werden.

Abbildg. 26.70 ASPX-Seite und XAP-Datei in der Ordnerliste des SharePoint



Silverlight-Anwendung testen

Zum Abschluss können Sie Ihre Anwendung noch testen.

1. Starten Sie den Browser mit der Adresse der »ListBox.aspx«.
2. Öffnen Sie ein weiteres Fenster und geben Sie ein neues Projekt in die Projektliste ein.
3. Wechseln Sie wieder zur ListBox und klicken Sie auf die Schaltfläche Liste aktualisieren.

Die Liste wird aktualisiert und Sie können erkennen, dass die neue Liste im Hintergrund angefordert wird. Die *.aspx*-Seite wird dabei nicht neu nachgeladen.

Zusammenfassung

Dieses Kapitel war sicherlich umfassend – auch wenn wir nicht annähernd das beschreiben konnten, was letztlich programmatisch in SharePoint möglich ist. Sie haben jedoch die verschiedenen Möglichkeiten der SharePoint-Programmierung anhand diverser Beispiele kennen gelernt, so dass Sie nun auf eine solide Basis zurückgreifen können, die Ihnen für den weiteren Wissensaufbau in diesem Bereich dienlich sein wird. Bleiben Sie dran – Programmierung macht Spaß!