PHP und Silverlight

Im Zwiegespräch

PHP ist eine Servertechnologie, Silverlight wird clientseitig ausgeführt. Wie man PHP und Silverlight trotzdem dazu bringt, miteinander in Dialog zu treten, zeigt dieser Workshop. **Von Jan Schenk**

Info

Auf einen Blick

- Der Workshop zeigt, wie PHP und Silverlight zusammenarbeiten können.
- Dargestellt werden an Beispielen Webservices via JSON sowie DOM-Manipulation.

Das brauchen Sie

- Installiertes Silverlight.
- Beispiellistings von www.phpjournal.de, Rubrik *Aktuelles Heft*.

Silverlight, Microsofts Rich Interactive Application Framework, wird, wenn es nicht gerade out of Browser läuft, per Plug-in im Browser ausgeführt. PHP hingegen wird als Server-Erweiterung für Webserver ausgeführt. Dies geschieht vor beziehungsweise während der Auslieferung einer Seite an den Anwender. Es gibt also für die einzelne Technologie erst einmal keinen Grund und auch keinen Weg, über die andere Bescheid zu wissen. So koexistieren Silverlight und PHP friedlich in demselben Revier, und für den Besucher einer Webseite ist von Verbrüderung oder gar Teamspiel nichts zu sehen oder zu spüren.

In datengetriebenen Anwendungen, beispielsweise mit Zugriffen auf eine serverseitige Datenbank, ist die Kommunikation vom und – im Fall von Silverlight – zurück zum Server essenziell. Oder es gibt die Notwendigkeit – im Fall von PHP – in die Rich Interactive Application hineinzukommunizieren, um der Applikation mitzuteilen, in welcher Umgebung mit welchen Templates sie zu arbeiten hat. Beides erfordert einen Brückenschlag von der Server- zur Clienttechnologie und wieder zurück. Der richtige und einfachste Weg dorthin sind sogenannte Webservices. Über eine standardisierte Schnittstelle spricht eine Client-Anwendung mit einer Server-Anwendung, nimmt Befehle entgegen oder gibt sie, erhält Daten oder sendet sie. In vielen Fällen ist dies der einzige Weg, um clientseitige Prozesse vom Server aus steuern zu können.

»Verwendung eines Webservices«

Beginnen wir mit dem denkbar einfachsten Fall für die Verwendung eines Webservices. Da Silverlight eine Client-Technologie ist, gibt es keinen Zugriff auf die genaue Serverzeit des auslieferndenden Servers, man muss sich vielmehr auf die Systemzeit des Clients verlassen. In vielen Anwendungen ist dieser Umstand schlicht und ergreifend nicht akzeptabel, da Manipulierbarkeit und Ungenauigkeit erhebliche Mängel darstellen würden.

Per »phpinfo()« lässt sich die Verfügbarkeit von JSON auf einem System überprüfen.

imap

IMAP c-Client Version	2004
SSL Support	enabled

json

json support	enabled
json version	1.2.1

libxml

libXML support	active
libXML Version	2.7.3
libXML streams	enabled

Wir beginnen mit der Exponierung der Serverzeit über einen JSON-Webservice (Javascript Object Notation) in PHP. Dank der JSON-Extension für PHP (php.net/json), die seit der PHP-Version 5.2 fester Bestandteil jedes offiziellen Builds ist, ist die Komponente schnell konstruiert. Davor empfiehlt es sich jedoch, sicherheitshalber per *<?php phpinfo() ?>* die Verfügbarkeit von JSON zu überprüfen.

Sollten Sie keine JSON-Unterstützung in der Ausgabe der Funktion *phpinfo()* finden, ist PHP ohne JSON-Unterstützung kompiliert worden. Haben Sie Ihre PHP-Installation selbst kompiliert, verzichten Sie beim Kompilieren auf den Parameter *--disable-json*.

Der PHP-Code für das Serverzeit-Experiment per JSON sieht so aus:

```
<?php
class ServerDateTime
{
public $ServerDT;
}

if(isset($_GET['ServerDateTime']))
{
$dt = new ServerDateTime();
$dt->ServerDT = date('Y-m-d H:i:s');
echo json_encode($dt);
}
?>
```

Dieser Code erzeugt im Browser folgende simple JSON-Ausgabe:

```
{"ServerDT":"2009-09-28 09:47:06"}
```

Man könnte für den PHP-Code auch ein wesentlich einfacheres klassenloses Konstrukt verwenden, allerdings soll der Code im Folgenden weiter ausgebaut werden. Deshalb haben wir uns schon jetzt für eine Klasse und die *if*-Abfrage als Basis entschieden.

Für den Silverlight-Code benötigen wir eine Präsentationsschicht in Form von Extensible-Application-Markup-Language-Code (XAML), dazugehörigem Code-Behind in C# (oder wahlweise VB.NET), und eine einfache C#-Klasse. Im XAML-Code beschränken wir uns auch im Weiteren auf eine schmucklose Anzeige in Form einfacher Textfelder, um den Code übersichtlich und leicht verständlich zu halten. Der künstlerischen Gestaltung von XAML-basierenden Oberflächen sind aber kaum Grenzen gesetzt. Wer ausprobieren möchte, wie leistungsfähig die von Silverlight verwendeten XAML-Oberflächen sind, sollte sich mit Expression Blend (www.microsoft.com/germany/expression) auseinandersetzen.

Der XAML-Code für die Datei *Page.xaml* wird im nachfolgenden Listing dargestellt:

```
<UserControl x:Class=
"SilverlightJsonWsSimple.Page"
xmlns="http://schemas.microsoft.com/
winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.
com/winfx/2006/xaml">
<Grid x:Name="LayoutRoot"
Background="White">
<TextBlock Height="21"
HorizontalAlignment="Left"
Name="textBlock1" Text="TextBlock"
VerticalAlignment="Top" Width="120"
/>
</Grid>
</UserControl>
```

Die Code-Behind-Datei *Page.xaml.cs* ist hingegen schon etwas komplizierter:

```
using System;
using System.Net;
using System.Windows.Controls;
using System.Json;
using System.Windows.Threading;

namespace SilverlightJsonWsSimple
{
public partial class Page :
UserControl
{
private long ServerTimeDiff;

public Page()
{
InitializeComponent();
GetServerTime();
CreateDispatcher();
}
private void CreateDispatcher()
{
DispatcherTimer timer =
new DispatcherTimer();
timer.Interval =
new TimeSpan(0, 0, 1);
timer.Tick += new
EventHandler(timer_Tick);
timer.Start();
}
void timer_Tick(object sender,
EventArgs e)
{
textBlock1.Text = DateTime.Now.
AddTicks(ServerTimeDiff).
TimeOfDay.ToString();
}

private void GetServerTime()
{
Uri serverUri = new Uri(
"http://localhost/php5.2.10/
index.php?ServerDateTime");
WebClient serverTimeClient =
new WebClient();
serverTimeClient.OpenReadCompleted
+=new OpenReadCompleted
EventHandler (serverTimeClient_
OpenReadCompleted);
serverTimeClient.OpenReadAsync
(serverUri);
}

void serverTimeClient_
OpenReadCompleted(object sender,
OpenReadCompletedEventArgs e)
{
using (e.Result)
{
JsonObject result =
(JsonObject)JsonObject.Load
(e.Result);
ServerTimeDiff =
DateTime.Parse(result["ServerDT"]).
Ticks - DateTime.Now.Ticks;
}
}
}
}
```

Autor

Jan Schenk ist Developer Evangelist bei Microsoft Deutschland. Dort ist er für die Bereiche Web und Live Services zuständig. Die Themen ASP.NET, Windows Azure, Silverlight und PHP Interoperability bilden seine Fokuspunkte. Jan Schenk hat acht Jahre Erfahrung im Free-Software-Umfeld gesammelt, bevor er 2008 zu Microsoft Deutschland wechselte. Er kennt die Vor- und Nachteile jeder Seite und spricht darüber gern offen.

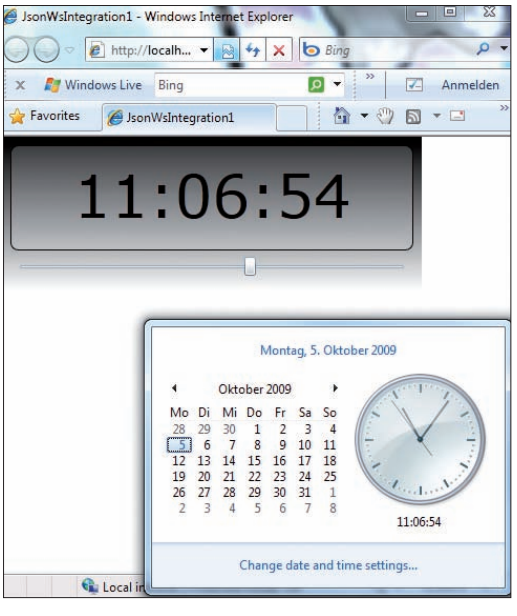
▶ **blogs.msdn.com/jansche**

Über die Methode *GetServerTime()* stellen wir eine Verbindung zum Webservice her. Die Methode ihrerseits ruft, sobald das Ergebnis geladen ist, das Event *serverTimeClient_OpenReadCompleted* auf, in dem die Differenz zwischen Serverzeit und Clientzeit in Ticks (100 Nanosekunden) berechnet und abgespeichert wird. Haben wir diese Differenz, so können wir mit der Clientzeit unter Aufrechnung der Serverzeitdifferenz das Textfeld mit der genauen Uhrzeit bestücken, was über den Dispatcher funktioniert. Dieser ist zusätzlich dafür zuständig, dass das User-Interface im Sekundentakt auf dem neuesten Stand gehalten wird.

Kürzere Zeitspannen sind selbstverständlich möglich und würden über eine kürzere Time-Span definiert, beispielsweise per *timer.Interval = new TimeSpan(0, 0, 0, 0, 40);* (für 40 Millisekunden, was in etwa 25 Frames pro Sekunde entspricht, also Video-Darstellung).

»Zweiseitige Kommunikation«

Bauen wir das Beispiel nun auf zweiseitige Kommunikation aus. Bisher wurden nur Daten vom Server an den Client übermittelt. Ein etwas komplexeres Beispiel soll nun zeigen, wie Personendaten zur Aktualisierung zum Client gesendet werden, und nach Bearbeitung wieder zurück zum Server gelangen. Die Personendaten werden beispielhaft in PHP erzeugt, in einer Real-World-Anwendung wäre hier über einen Filter eine Auswahl an Daten aus einer Datenbank extrahiert worden. Für unser Beispiel erweitern wir das vorherige PHP-Skript um eine neue Klasse, eine Funktion und drei Blinddatensätze:



Auf dem Entwicklungssystem stimmen die Uhren selbstverständlich überein.

```
<?php
class ServerDateTime
{
public $ServerDT;
}

class Individual
{
public $FirstName;
public $LastName;
public $Email;
public $Position;
}
// dummy data generator start
$i1 = new Individual();
```

```
$i1->FirstName = "Franz";
$i1->LastName = "Huckler";
$i1->Email = "fh@bauernhof.de";
$i1->Position = utf8_encode
("Mähdrescherfahrer");

$i2 = new Individual();
$i2->FirstName = "Wolfgang";
$i2->LastName = "Dreschel";
$i2->Email = "wd@bauernhof.de";
$i2->Position = "Pflugfahrer";

$i3 = new Individual();
$i3->FirstName = "Thomas";
$i3->LastName = "Rosin";
$i3->Email = "tr@bauernhof.de";
$i3->Position = "Heuwender";
// dummy data generator stop

if(isset($_GET['ServerDateTime']))
{
$dt = new ServerDateTime();
$dt->ServerDT = date('Y-m-d H:i:s');
echo json_encode($dt);
}

if(isset($_GET['AllIndividuals']))
{
$return = array($i1, $i2, $i3);
echo json_encode($return);
}
?>
```

Hier wäre eine Besonderheit zu erwähnen, nämlich die Umwandlung von Umlauten in UTF8-konforme Strings:

```
$i1->Position = utf8_encode
("Mähdrescherfahrer");
```

Die Funktion *json_encode()* erwartet UTF8-Datensätze; Umlaute und andere Sonderzeichen des

Listing 1

Code-Behind-Datei: Personendaten

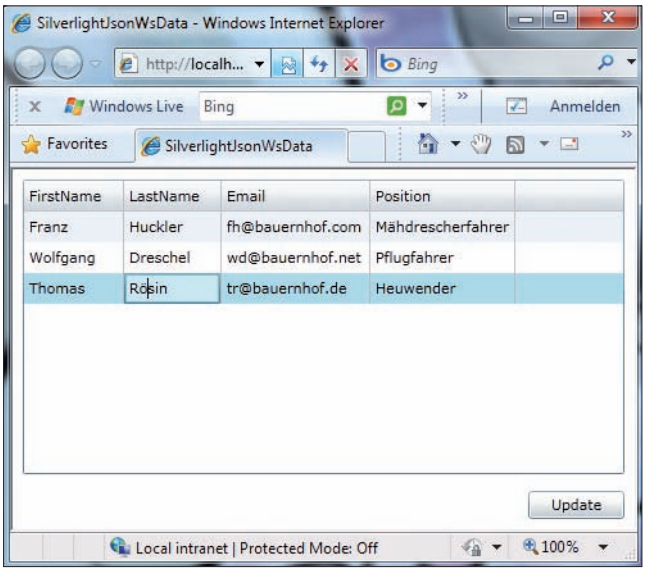
```
using System;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Json;
using System.Collections.Object
Model;
using System.IO;
using System.Text;
using
System.Runtime.Serialization.Json;

namespace SilverlightJsonWsData
{
public partial class Page :
UserControl
{
private ObservableCollection<
DataResult> dataCollection = new
ObservableCollection<DataResult>();

public Page()
{
InitializeComponent();

GetServerData();
}

private void GetServerData()
{
Uri serverUri = new Uri(
"http://localhost/php5.2.10/
index.php?AllIndividuals");
WebClient serverIndividualsClient =
new WebClient();
serverIndividualsClient.
OpenReadCompleted +=
new OpenReadCompletedEventHandler
(serverIndividualsClient_
OpenReadCompleted);
serverIndividualsClient.
OpenReadAsync(serverUri);
}

void serverIndividualsClient_
OpenReadCompleted(object sender,
OpenReadCompletedEventArgs e)
{
using (e.Result)
{
JsonArray resultArray =
(JsonArray)JsonArray.Load(e.Result);

foreach (JsonObject RawObject in
resultArray)

{
DataResult dr = new DataResult();
dr.FirstName =
RawObject["FirstName"];
dr.LastName =
RawObject["LastName"];
dr.Email =
RawObject["Email"];
dr.Position =
RawObject["Position"];

dataCollection.Add(dr);
}

DetailsView.ItemsSource =
dataCollection;
}
}

private void Button_Click
(object sender, RoutedEventArgs e)
{

Uri serverUri = new Uri
("http://localhost/php5.2.10/
sendIndividuals.php");
DataContractJsonSerializer ser =
new DataContractJsonSerializer
(typeof(ObservableCollection<
DataResult>));
MemoryStream mem = new
MemoryStream();
ser.WriteObject
(mem,dataCollection);
string serdata =
Encoding.UTF8.GetString
(mem.ToArray(), 0,
(int)mem.Length);

WebClient backPostClient =
new WebClient();
backPostClient.UploadStringCompleted
+= new
UploadStringCompletedEventHandler
(cnt_UploadStringCompleted);
backPostClient.Headers
["Content-type"] =
"application/json";
backPostClient.Encoding =
Encoding.UTF8;
backPostClient.UploadStringAsync
(serverUri, "POST", serdata);
}

void cnt_UploadStringCompleted
(object sender,
UploadStringCompletedEventArgs e)
{
if (e.Result != null)
{
      SubmitButton.Content =
      "Sent!";
}
}

}
}
```

ISO-8859-1-Zeichensatzes führen zu einem Fehler. Silverlight-seitig empfangen wir die Daten und verpacken sie editierbar in ein Data Grid:

```
<UserControl
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"
xmlns:data="clr-
namespace:System.Windows.Controls;
assembly=
System.Windows.Controls.Data"
x:Class="SilverlightJsonWsData.Page"
d:DesignHeight="300"
d:DesignWidth="400">

<Grid x:Name="LayoutRoot"
Background="White">

<data:DataGrid x:Name="DetailsView"
Margin="8,8,8,43"/>
<Button x:Name="SubmitButton"
HorizontalAlignment="Right"
VerticalAlignment="Bottom"
Width="75" Content="Update"
Margin="0,0,8,8"
Click="Button_Click" />

</Grid>
</UserControl>
```

Die zugehörige Code-Behind-Datei (Listing 1) kümmert sich um die Business-Logik wie Datenabruf (*GetServerData()*-Methode), Datenaufbereitung (*foreach (JsonObject RawObject in resultArray)*), Datenserialisierung und -rücksendung (im *Button_Click()*-Event). Die verwendete Klasse *DataResult* definiert sich wie folgt:

```
namespace SilverlightJsonWsData
{
public class DataResult
{
public string FirstName {
get; set;
}
public string LastName { get; set; }
public string Email { get; set; }
public string Position { get; set; }
}
}
```



Die drei Beispieldatensätze werden in unserem Frontend angezeigt und können editiert werden

Um Daten von der Silverlight-Applikation zu empfangen, verwenden wir ein weiteres PHP-Skript (*sendIndividuals.php*), das unsere Dummy-Daten in eine CSV-Datei speichert, die man vorab händisch erstellen und mit den passenden Rechten versehen sollte:

```
<?php
$xml =
file_get_contents
('php://input');
if($xml != "")
{
$obj =
json_decode($xml,
true);
$dumpFile = "dump.csv";
$fh = fopen($dumpFile, 'a') or
die("can't open file");

foreach($obj as $item)
{
$FirstName =
filter_var($item['FirstName'],
FILTER_SANITIZE_STRING);
$LastName = filter_var($item
['LastName'],
FILTER_SANITIZE_STRING);
$Email = filter_var($item['Email'],
FILTER_VALIDATE_EMAIL);
$Position = filter_var($item
['Position'],
FILTER_SANITIZE_STRING);
fwrite($fh, $FirstName . ";" .
$LastName . ";" . $Email . ";" .
$Position . ";\n");
}
fclose($fh);
}
else
echo "no data POSTed";
?>
```

Unbedingt sollte auf dem Server, wie im Beispiel oben, noch einmal gefiltert und validiert werden. Die PHP-Extension *filter* bietet dafür hervorragende Möglichkeiten, die im PHP-Manual unter php.net/manual/de/book.filter.php detailliert beschrieben sind.

Wie bei Clienttechnologien üblich sind die übertragenen Daten nicht sicher und können bösartige Code-Injection-Versuche enthalten. Serverseitige Filtervorgänge sind also keine Verschwendung von Ressourcen, sondern absolut erforderlich. ▶

Info

Werkzeuge

Kostenlose Werkzeuge für das Entwickeln mit Silverlight finden Sie in der folgenden Zusammenstellung:

Visual Web Developer Express Edition – die Entwicklungsumgebung
▸ **www.microsoft.com/germany/express**

Die Silverlight-Entwickler-Tools – diese installieren die Developer Runtime(Debugging) anstelle des normalen Plug-ins ohne Debugging
▸ **go.microsoft.com/fwlink/?LinkID=143571**

Das Silverlight Toolkit – weitere Komponenten, die die Erstellung von Silverlight-Anwendungen vereinfachen
▸ **go.microsoft.com/fwlink/?LinkID=157133**

Hilfreiche Tools und Anleitungen, sowie eine detaillierte Beschreibung zur Installation der Grundvoraussetzungen finden Sie unter
▸ **silverlight.net/getstarted**

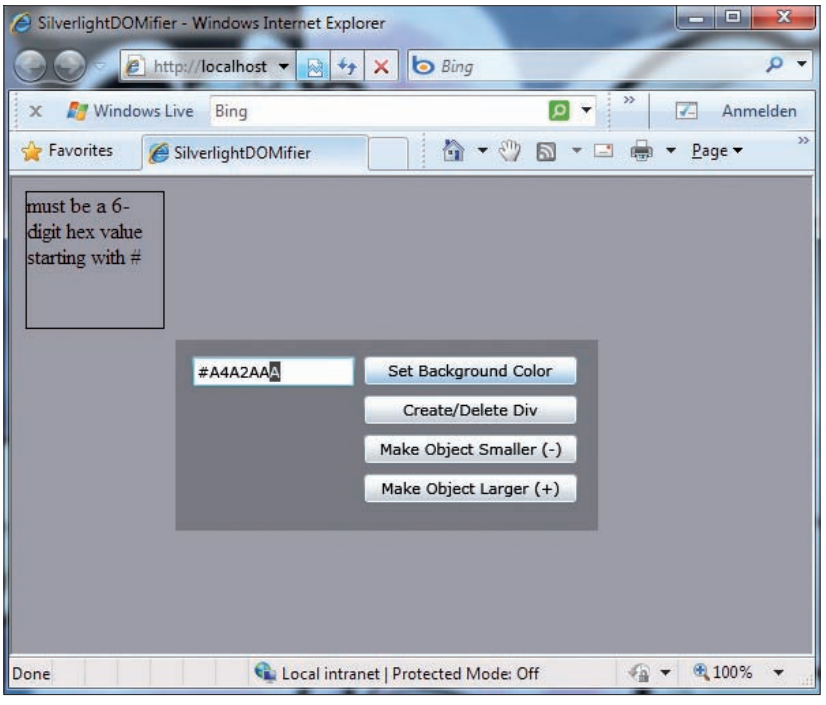
Optional: Expression Blend 3 – die komfortable Design-Umgebung (kostenlose Trial verfügbar)
▸ **www.microsoft.com/germany/expression**

Haben wir nun alle Codebausteine zusammengefügt, lässt sich die Applikation mit dem Aufruf der Silverlight-Komponente starten. Die drei Beispieldatensätze werden in unserem Frontend angezeigt und können editiert werden. Änderungen müssen im Data Grid nicht, wie zum Beispiel durch einen Wechsel des Auswahlfelds, bestätigt werden. Zurück an den Server gelangen die Daten über das empfangende PHP-Skript *sendIndividuals.php*, sobald der Update-Button betätigt wird.

Nach erfolgreichem Aktualisieren der Daten im Frontend schreibt *sendIndividuals.php* die übertragenen Daten stellvertretend für eine Datenbankoperation in eine Datei *dump.csv:*

```
Franz;Huckler;fh@bauernhof.com;
Mähdrescherfahrer;
Wolfgang;Dreschel;wd@bauernhof.net;
Pflugfahrer;
Thomas;Rösin;tr@bauernhof.de;Heuwender;
```

Wenn man weiß, wo man anfangen muss, sind PHP-Webservices à la JSON mit Silverlight recht einfach zu bewerkstelligen. Die beidseitige Akzeptanz eines Standards, in unserem Fall JSON, zeigt, dass man die grenzenlosen Möglichkeiten des Internets nicht in nur einem Technologie-Stack leben muss und Kenntnisse in zwei oder mehr dieser Stacks hervorragend unter einen Hut zu bringen sind. Interoperabilität ist eben mehr als nur ein Modewort.



Mit Silverlight lassen sich ganze HTML-Objekte hinzufügen, modifizieren oder löschen.

»Alternative: DOM-Manipulation«

Webservices sind aber nicht immer notwendig. Denn es gibt auch andere Wege, um zwei grundverschiedenen Technologien wie PHP und Silverlight an einem Strang ziehen zu lassen. Zwar sind diese Wege oft sehr viel weniger flexibel. Diese haben aber durchaus ihre Daseinsberechtigung und sollen anhand des Beispiels HTML-DOM-Manipulation kurz vorgestellt werden.

Ein Merkmal der Servertechnologie PHP ist, dass Modifikationen am HTML-Document-Object-Model (DOM) ohne Einschränkungen vorgenommen werden können – und ohne dass der Anwender davon etwas merkt. Das statische HTML-Grundgerüst einer Seite wird durch PHP-Bausteine, die ihrerseits dynamisch HTML-Code erzeugen, komplettiert.

Aber auch Silverlight hat die Möglichkeit, auf das DOM zuzugreifen und Werte zu lesen oder auch zu verändern. Die Fähigkeit, ganze HTML-Objekte hinzuzufügen, zu modifizieren oder zu löschen ist in den Basisbibliotheken von Silverlight seit Version 2 enthalten, wie dieses einfache Beispiel zeigt:

```
<UserControl x:Class=
"SilverlightDOMifier.Page"
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"
Background="{x:Null}"
d:DesignHeight="131" d:DesignWidth="300"
BorderBrush="#FF404040">

<Grid x:Name="LayoutRoot"
Background="#83575757"
OpacityMask="Black">
<TextBox Height="23"
HorizontalAlignment="Left"
Margin="12,12,0,0" Name="textBox1"
VerticalAlignment="Top" Width="120"
Text="#FFFFFF" />
<Button Content="Set Background Color"
Margin="138,12,15,0" Name="button1"
VerticalAlignment="Top"
Click="button1_Click" />
<Button Content="Create/Delete Div"
Margin="138,41,15,0" Name="button2"
VerticalAlignment="Top"
Click="button2_Click" />
<Button Content="Make Object Smaller (-)"
Margin="138,70,15,0" Name="button3"
VerticalAlignment="Top"
Click="button3_Click" />
<Button Content="Make Object Larger (+)"
Margin="138,99,15,0" Name="button4"
VerticalAlignment="Top"
Click="button4_Click" />
</Grid>
</UserControl>
```

Die zugehörige Code-Behind-Datei zeigt Listing 2. Wir bleiben mit unseren Operationen hier natürlich clientseitig. Für viele Anwendungsbereiche, wie zum Beispiel hybride Webseiten, die sowohl klassisches HTML als auch reichhaltige Clienttechnologien, nämlich Silverlight, beinhal-

ten, bietet die gemeinsame HTML-DOM-Modifikation vielfältige Möglichkeiten. Aber auch aus dem PHP-Code heraus können auf eine wirkungsvolle und dennoch sehr einfache Art und Weise Silverlight-Anwendungen gesteuert werden.

Um mit PHP Einfluss auf Silverlight zu nehmen, muss man als Entwickler weder Webservices nutzen noch mit Silverlight DOM-Manipulation vornehmen. Silverlight liest prinzipiell die Parameter, die das einbettende *object*-Tag liefert, ein. So weiß die Silverlight-Applikation von Haus aus über bestimmte Zustände, die im *object*-Tag definiert wurden, selbst Bescheid. Diese Parameter lassen sich auf Wunsch erweitern und anpassen, so dass eine dynamische, einseitige Kommunikation von PHP zu Silverlight möglich ist.

Indem wir in dem HTML-Dokument das *object*-Tag, das die Silverlight-Applikation einbettet, einige Parameter durch PHP dynamisch erstellen lassen, nehmen wir Einfluss auf die Lokalisierung der Silverlight-Applikation: ▶

Listing 2

Code-Behind-Datei: DOM-Manipulation

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Browser;

namespace SilverlightDOMifier
{
  public partial class Page :
  UserControl
  {
    public Page()
    {
      InitializeComponent();
    }

    private void button1_Click
    (object sender,
    RoutedEventArgs e)
    {
      if (textBox1.Text.Length == 7
      && textBox1.Text.Substring(0,
      1) == "#")

      {
        string bgValue =
        textBox1.Text;
        HtmlPage.Document.
        SetProperty("bgcolor",
        bgValue);
      }
      else

      {
      if (HtmlPage.Document.
      GetElementById("newDiv")
      == null)
      CreateDiv();
      HtmlElement errorDiv =
      HtmlPage.Document.
      GetElementById("newDiv");
      errorDiv.SetProperty
      ("innerHtml", "must be a
      6-digit hex value starting
      with #");
      }
    }

private void button2_Click
(object sender, RoutedEventArgs e)
{
    if (HtmlPage.Document.
    GetElementById("newDiv") == null)

    {
    CreateDiv();
    }
    else
    {
    HtmlElement newDiv =
    HtmlPage.Document.GetElementById
    ("newDiv");
    HtmlPage.Document.Body.
    RemoveChild(newDiv);
    }
    }

private void button3_Click(object
sender, RoutedEventArgs e)
    {
    HtmlElement embedObject =
    HtmlPage.Document.GetElementById
    ("slEmbedObject");
    int newObjectWidth =
    Convert.ToInt32
    (embedObject.GetAttribute
    ("width")) - 10;
    int newObjectHeight =
    Convert.ToInt32
    (embedObject.GetAttribute
    ("height")) - 10;
    embedObject.SetAttribute
    ("width",
    newObjectWidth.ToString());
    embedObject.SetAttribute
    ("height",
    newObjectHeight.ToString());
    }

private void button4_Click(object
sender, RoutedEventArgs e)
    {
    HtmlElement embedObject =
    HtmlPage.Document.GetElementById
    ("slEmbedObject");
    int newObjectWidth =
    Convert.ToInt32(embedObject.
    GetAttribute ("width")) + 10;
    int newObjectHeight =
    Convert.ToInt32
    (embedObject.GetAttribute
    ("height")) + 10;
    embedObject.SetAttribute
    ("width",
    newObjectWidth.ToString());
    embedObject.SetAttribute
    ("height",
    newObjectHeight.ToString());
    }

private static void CreateDiv()
    {
    HtmlElement newDiv =
    HtmlPage.Document.CreateElement
    ("div");

    newDiv.SetProperty
    ("innerHTML", "neues Div");
    newDiv.SetAttribute("id",
    "newDiv");

    newDiv.SetStyleAttribute(
    "background-color",
    "#999999");
    newDiv.SetStyleAttribute
    ("border",
    "solid 1px black");
    newDiv.SetStyleAttribute
    ("position",
    "absolute");
    newDiv.SetStyleAttribute
    ("top", "10px");
    newDiv.SetStyleAttribute
    ("left", "10px");
    newDiv.SetStyleAttribute
    ("width", "100px");
    newDiv.SetStyleAttribute
    ("height", "100px");

    HtmlPage.Document.Body.
    AppendChild(newDiv);
    }

    }
}
```

```
<object data=
"data:application/x-silverlight-2,"
type="application/x-silverlight-2"
width="100%" height="100%">
<param name="source" value=
"ClientBin/
SilverlightReadObjectParams.xap" />
<param name="onerror" value=
"onSilverlightError" />
<param name="background"
value="white" />
<param name="minRuntimeVersion"
value="3.0.40818.0" />
<param name="autoUpgrade" value="true" />
<param name="Culture" value="
<?php echo 'de-DE'?>" />
<param name="UICulture" value="
<?php echo 'de-DE'?>" />
<param name="initParams"
value="key1=value1,key2=value2,
<?php echo 'surname=Jan,name=Schenk'?>"
/>
<a href="http://
go.microsoft.com/fwlink/?LinkID=124807"
style="text-decoration: none;">
<img src="http://
go.microsoft.com/fwlink/?LinkId=108181"
alt= "Get Microsoft Silverlight"
style="border-style: none"/>
</a>
</object>
```

Auf diese gesetzten Werte kann man in der Silverlight-Applikation per *Thread.CurrentThread.CurrentCulture* und *Thread.CurrentThread.CurrentUICulture* zugreifen. Man kann aber noch weiter gehen und Silverlight Key-Value-Paare übergeben, ohne einen Schlüssel zu benutzen, der Silverlight bereits bekannt ist. Dafür benutzt man den Parameternamen *initParams* nach dem Muster:

```
<param name="initParams"
value="key1=value1,key2=value2" />.
```

In der *App.xaml.cs* wird Silverlight angewiesen, wie es die übergebenen Paare behandeln soll. In die bereits existierende Methode *Application_Startup* wird unter *this.RootVisual = new Page();* dieser Code hinzugefügt:

```
if (e.InitParams != null)
{
foreach (var data in e.InitParams)
{
this.Resources.Add(data.Key, data.Value);
}
}
```

Da der Zugriff auf die *StartupEventArgs* nur in der *App.xaml.cs* und dort nur in der Methode *Application_Startup* möglich ist, ist es zwingend erforderlich, dass die *foreach*-Schleife dort ausgeführt wird.

Nun kann an beliebiger Stelle im Code, zum Beispiel innerhalb der *Page.xaml.cs,* auf diese Ressourcen zugegriffen werden:

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Threading;

namespace SilverlightReadObjectParams
{
public partial class Page : UserControl
{
public Page()
{
InitializeComponent();
this.Loaded += new
RoutedEventHandler(Page_Loaded);
}
void Page_Loaded(object sender,
RoutedEventArgs e)
{
string curCulture =
Thread.CurrentThread.
CurrentCulture.ToString();
string curUICulture =
Thread.CurrentThread.
CurrentUICulture.ToString();
textBlock1.Text = curCulture + " : " +
curUICulture;
```

Es gibt eine Reihe kostenloser Werkzeuge für das Entwickeln mit Silverlight.

```
string fillBox = "";
if (App.Current.Resources.Contains
("key1"))
fillBox += "key1 : " +
App.Current.Resources["key1"].ToString()
+ Environment.NewLine;
if
(App.Current.Resources.Contains("key2"))
fillBox += "key2 : " +
App.Current.Resources["key2"].ToString()
+ Environment.NewLine;
textBlock2.Text = fillBox;
}
}
}
```

Prinzipiell lassen sich Silverlight-einbettende Seiten in zwei Kategorien aufteilen: reine Silverlight-Seiten, bei denen HTML-Code und Browser nur zur Auslieferung dienen, und klassischere, auf HTML basierende Seiten, die mit einzelnen Silverlight-Objekten angereichert sind.

Viele Entwickler von RIA-Anwendungen werden die zweite Methode vorziehen. Zwar sind, speziell in Silverlight, viele Probleme inzwischen ausgemerzt, die früher für Rich-Interactive-only-Anwendungen gegolten haben: fehlende Deep-Links und Bookmark-Möglichkeiten, schlechte Suchmaschinenindizierung, schlechte Barrierefreiheit. Dennoch ist es sinnvoll, eine Webanwendung nur mit so viel RIA zu versehen wie notwendig. Damit erhöht man die Wartbarkeit und Übersichtlichkeit, und man optimiert den Nutzen für den Anwender.

Alle obigen Code-Beispiele können im Übrigen mit kostenlos verfügbaren Werkzeugen gebaut und kompiliert werden. Von einigen gibt es auch kostenpflichtige Versionen, deren Funktionsumfang erheblich von dem der kostenfreien Varianten abweichen kann (siehe Kasten auf Seite 31).

»Cross-Domain-Zugriffsschutz«

Wenn der Silverlight-Server nicht der gleiche Server ist, wie der, der den PHP-Webservice hostet, gilt zu beachten, dass man, selbst bei lokaler Ausführung des Projekts, auf eine *crossdomain.xml*-Datei im Dokumentenstammverzeichnis des Webservers angewiesen ist, die folgenden Inhalt hat:

```
<?xml version="1.0"?>
<cross-domain-policy>
<allow-http-request-
headers-from domain="*"
headers="*"/>
</cross-domain-policy>
```

Wir erlauben damit den Zugriff von anderen Quellen als dem Webservice-Server. Gleiches gilt bei Veröffentlichung des Projekts. Hier ist es aber sinnvoll, den *domain*-Parameter auf die Domain anzupassen, die das Silverlight-Projekt hostet. Zu beachten ist, dass Cross-Domain-Richtlinien keinen Zugriffsschutz und keinerlei Sicherheitskonzept bieten, da die Interpretation der *crossdomain.xml* beim Client liegt und umgangen werden kann. **[mb]**