<http://msdn2.microsoft.com/en-us/magazine/cc135987.aspx>

Ekstremalne programowanie w ASP.NET

Jak wbudować Silverlight w kontrolki ASP.NET

Fritz Onion

Kod dostępny do pobrania: [ExtremeASPNET2008\_01.exe](http://download.microsoft.com/download/f/2/7/f279e71e-efb0-4155-873d-5554a0608523/ExtremeASPNET2008_01.exe) (700 KB)
[Przejrzyj kod w trybie on-line](http://msdn2.microsoft.com/en-us/magazine/cc164181.aspx)

 Spis treści

[Stosowanie Silverlight](#_Toc198369833)

[Utworzenie własnej kontrolki Silverlight](#_Toc198369834)

[ASP.NET AJAX i IScriptControl](#_Toc198369835)

[Kontrolki asp:Xaml i asp:Media](#_Toc198369836)

Wielu programistów ASP.NET na całym świecie zastanawia się, w jaki sposób i gdzie wykorzystać Silverlight™ w swoich aplikacjach. Odpowiedź nie jest prosta, ponieważ istnieje wiele możliwych rozwiązań. Kontrolka Silverlight może być jedynym elementem strony, zawierającym całą jej treść. Inne, bardziej prawdopodobne rozwiązanie, to wybranie fragmentów strony, które najbardziej zyskają na wzbogaceniu ich z zastosowaniem Silverlight, wykorzystanie zaawansowanych elementów interfejsu użytkownika tam, gdzie są najbardziej potrzebne i pozostawienie całej struktury aplikacji bez większych zmian.

To drugie podejście jest czasem nazywane uzupełnianiem strony o „wyspy bogactwa”. Dzięki Silverlight, wyspy te mogą współpracować z otaczającą je treścią z wykorzystaniem skryptów i procedur obsługi zdarzeń.

W artykule tym omówię techniki budowy własnych kontrolek serwerowych zawierających elementy Silverlight. Wbudowanie treści Silverlight we własne kontrolki ASP.NET ma kilka zalet, które sprawiają, że jest to bardzo interesująca technika. Ten sposób integracji Silverlight jest równie łatwy, jak stosowanie kontrolek serwerowych, co znacznie podnosi szanse popularyzacji tej techniki. Sposób tworzenia procedur obsługi zdarzeń, ustawiania właściwości i wywoływania metod kontrolki Silverlight jest taki sam, jak każdej innej kontrolki. Podejście to pozwala także uniknąć zaśmiecania strony kodem JavaScript, potrzebnym do obsługi Silverlight, co z kolei upraszcza wdrażanie i konserwowanie budowanych w ten sposób aplikacji.

# Stosowanie Silverlight

Zanim opiszę sposób utworzenia własnej kontrolki zawierającej treść Silverlight, chciałbym omówić elementy niezbędne do wyświetlenia treści Silverlight. Pokażę, jak osadzić plik XAML w stronie ASP.NET i obsługiwać interakcje pomiędzy stroną a treścią Silverlight. Mając gotową stronę o funkcjonalności takiej, jaką chcę umieścić w kontrolce, łatwiej będzie zrozumieć, jak należy zbudować kontrolkę. Zaczniemy od utworzenia pliku XAML, który będzie wyświetlany z wykorzystaniem Silverlight. Chciałbym, by kod XAML był na tyle prosty, by nie utrudniał prezentacji kontrolki, a jednocześnie na tyle złożony, by był interesujący. Dlatego zdecydowałem się na wyświetlenie kuli i tytułu. Plik XAML przedstawiony na **ilustracji 1.** definiuje element TextBlock, który posłuży do wyświetlenia tytułu, i element Ellipse wypełniony promienistym przejściem tonalnym (radial gradient). W pliku, w części Resources elementu Canvas, zdefiniowano także dwa elementy Storyboard, opisujące dwie animacje powodujące odpowiednio powiększenie i zmniejszenie kuli w czasie dwóch sekund. Plik XAML chcę wyświetlać za pomocą Silverlight i — z wykorzystaniem procedur obsługi zdarzeń napisanych w JavaScript — w odpowiedzi na kliknięcia kuli ustawiać tekst tytułu i uruchamiać animacje.

 Ilustracja 1. Plik Sphere.xaml

Kopiuj kod

<!-- File: Sphere.xaml -->

<Canvas

 xmlns="http://schemas.microsoft.com/client/2007"

 xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

 Width="300" Height="300"

 Background="White"

 >

 <Canvas.Resources>

 <Storyboard x:Name="growAnimation">

 <DoubleAnimation

 Storyboard.TargetName="ellipse"

 Storyboard.TargetProperty="(Ellipse.Width)"

 To="250" Duration="0:0:2" />

 <DoubleAnimation

 Storyboard.TargetName="ellipse"

 Storyboard.TargetProperty="(Ellipse.Height)"

 To="250" Duration="0:0:2" />

 </Storyboard>

 <Storyboard x:Name="shrinkAnimation">

 <DoubleAnimation

 Storyboard.TargetName="ellipse"

 Storyboard.TargetProperty="(Ellipse.Width)"

 To="200" Duration="0:0:2" />

 <DoubleAnimation

 Storyboard.TargetName="ellipse"

 Storyboard.TargetProperty="(Ellipse.Height)"

 To="200" Duration="0:0:2" />

 </Storyboard>

 </Canvas.Resources>

 <TextBlock x:Name="titleText" Width="200" Height="24"

 Canvas.Left="94" Text="[Title]" TextWrapping="Wrap" />

 <Ellipse Width="200" Height="200"

 x:Name="ellipse" Canvas.Left="47" Canvas.Top="41">

 <Ellipse.Fill>

 <RadialGradientBrush GradientOrigin="0.75,0.25"

 Center="0.5,0.5" RadiusX="0.5" RadiusY="0.5">

 <GradientStop Color="Yellow" Offset="0" />

 <GradientStop Color="Green" Offset="1" />

 </RadialGradientBrush>

 </Ellipse.Fill>

 </Ellipse>

</Canvas>

Aby wyświetlić ten plik XAML na stronie internetowej, należy najpierw utworzyć wtyczkę Silverlight. Najprościej można to zrobić, dodając do strony referencję do pliku Silverlight.js, wchodzącego w skład zestawu SDK Silverlight 1.0. W pliku tym zdefiniowana jest metoda o nazwie Silverlight.createObject, którą można wykorzystać do utworzenia wtyczki. Wtyczka Silverlight musi być związana z jakimś elementem HTML (zwykle <div>), dlatego popularnym sposobem utworzenia wtyczki jest zdefiniowanie elementu <div> i umieszczenie w nim bloku skryptu z wywołaniem metody createObject.

**Na ilustracji 2. przedstawiono sposób utworzenia instancji kontrolki Silverlight i powiązania jej z elementem <div>. Założono, że** przedstawiony na **ilustracji 1.** **plik** Sphere.xaml znajduje się w tym samym katalogu, co plik z kodem strony. Strona załadowana w przeglądarce powinna wyświetlić piękną zieloną kulkę, widoczną na **ilustracji 3**.

 Ilustracja 2. Utworzenie kontrolki Silverlight i powiązanie jej z elementem <div>

Kopiuj kod

<%@ Page Language="C#" %>

<html >

<head runat="server">

 <script type="text/javascript" src="silverlight.js"></script>

</head>

<body>

 <form id="form1" runat="server">

 <div id="slControlHost">

 <script type="text/javascript">

 Silverlight.createObject(

 "Sphere.xaml",

 document.getElementById('slControlHost'), "slControl",

 { width:'300', height:'300', version:'1.0' },

 { onError:null, onLoad:null },

 null);

 </script>

 </div>

 </form>

</body>

</html>



Ilustracja 3. **Kontrolka wyświetlająca zieloną kulkę**

Następnym zadaniem jest uruchomienie interakcji z XAML, co pozwoli na programistyczne ustawianie tekstu tytułu i inicjowanie opisanych wcześniej animacji powiększających i zmniejszających kulę. Najprostszym sposobem realizacji tego zadania jest dodanie deklaracji przypisujących procedury obsługi zdarzeń do obiektów zdefiniowanych w pliku XAML. W naszym przykładzie animacje powiększenia i pomniejszenia inicjowane są w odpowiedzi na kliknięcie kuli przez użytkownika. To, która animacja powinna zostać uruchomiona, zależy od aktualnej wielkości kuli.

Najpierw zadeklarowałem procedurę obsługi zdarzenia MouseLeftButtonDown elementu Ellipse definiującego kulę, a następnie utworzyłem odpowiednią metodę w języku JavaScript. W analogiczny sposób utworzyłem procedurę obsługi zdarzenia Loaded elementu TextBlock, co pozwoli na ustawienie tekstu wyświetlanego ponad kulą. Deklaracje dodane do kodu XAML mają następującą postać:

Kopiuj kod

<Ellipse MouseLeftButtonDown="javascript:onSphereButtonDown" ...

<TextBlock Loaded="javascript:onTextLoaded" ...

Natomiast odpowiadające im metody JavaScript, umieszczone w nowym bloku skryptu na stronie ASP.NET, przedstawione są na **ilustracji 4**.

 Ilustracja 4. Metody JavaScript w nowym bloku skryptu

Kopiuj kod

<script type="text/javascript">

 function onSphereButtonDown(sender, args)

 {

 // Run the grow or shrink animation,

 // depending on whether it is currently

 // 'grown' or 'shrunk'

 var animationName = (sender.Width==200) ?

 "growAnimation" : "shrinkAnimation";

 var sl = sender.getHost();

 var animation = sl.content.findName(animationName);

 if (animation)

 animation.begin();

 }

 function onTextLoaded(sender, args)

 {

 sender.Text = "My Growing Sphere";

 }

</script>

To wszystko, co potrzebne jest do ręcznego osadzenia treści Silverlight w stronie ASP.NET. Gdy strona ta zostanie wyświetlona w przeglądarce, tytuł będzie zawierał tekst „My Growing Sphere”. Gdy użytkownik kliknie kulę po raz pierwszy, kula powiększy się. Drugie kliknięcie spowoduje zmniejszenie kuli. Jak widać, integracja treści Silverlight ze stroną ASP.NET jest dość prosta. Łatwe jest też modyfikowanie treści Silverlight, ponieważ cały model obiektowy jest dostępny dla skryptów na stronie.

W przypadku prostych elementów, takich jak nasza kula, można przygotować opakowanie w postaci własnej kontrolki, którą można łatwo wykorzystać w aplikacji poprzez proste przeciągnięcie jej na obszar projektowanej strony. Właśnie w tym celu można wykorzystać technikę opakowywania Silverlight.

# Utworzenie własnej kontrolki Silverlight

Zadanie osadzenia Silverlight w kontrolce zacznę od utworzenia nowej kontrolki ASP.NET:

Kopiuj kod

namespace MsdnMagazine

{

 public class SilverlightSphere : Control

 {

 }

}

W tym przypadku zdecydowałem się na dziedziczenie po System.Web.UI.Control. Nie chcę dziedziczyć po WebControl, ponieważ dodatkowe właściwości stylu, definiowane przez tę klasę, nie mają zastosowania do naszej kontrolki.

Teraz musimy zająć się funkcjami obsługi zdarzeń, które poprzednio napisałem ręcznie. Najprostszym sposobem dodania metod JavaScript do kontrolki ASP.NET jest utworzenie pliku JavaScript i dodanie go jako zasobu do skompilowanego podzespołu. Dwie utworzone wcześniej procedury obsługi zdarzeń przeniosłem do osobnego pliku JavaScript, nazwanego SilverlightSphere.js:

Kopiuj kod

 // plik: SilverlightSphere.js

 function onSphereButtonDown(sender, args)

 {

 // implementacja identyczna jak poprzednio

 }

 function onTextLoaded(sender, args)

 {

 // implementacja identyczna jak poprzednio

 }

Gotowy plik SilverlightSphere.js oraz oryginalny plik Silverlight.js dodałem jako zasoby do projektu kontrolki. Dzięki temu kontrolka może zostać zainstalowana jako pojedynczy podzespół niewymagający żadnych dodatkowych plików. Plik XAML także można umieścić w projekcie jako zasób i odwoływać się do niego oraz do dwóch plików JavaScript, korzystając z mechanizmu WebResource.axd, pozwalającego na ekstrakcję osadzonych zasobów w ASP.NET.

Po ustawieniu w Visual Studio® właściwości Build Action wszystkich trzech plików na Embedded Resource (przełącznik /res w przypadku kompilacji z wiersza polecenia), użyłem działającego na poziomie podzespołu atrybutu System.Web.UI.WebResource w celu nadania uprawnień pozwalających na obsługę tych zasobów za pomocą mechanizmu WebResource.axd i przypisania im typu MIME. Użyłem następujących deklaracji (SlSphere to nazwa projektu kontrolki):

Kopiuj kod

[assembly: WebResource("SlSphere.SilverlightSphere.js",

 "text/javascript")]

[assembly: WebResource("SlSphere.Silverlight.js",

 "text/javascript")]

[assembly: WebResource("SlSphere.Sphere.xaml", "text/xml")]

Pliki JavaScript są teraz wkompilowywane w podzespół jako osadzone zasoby. By umieścić referencje do tych plików w nagłówku strony, można skorzystać z metody RegisterClientScriptResource klasy ClientScriptManager (dostępnej jako Page.ClientScriptManager). Wywołania tej metody dla poszczególnych plików umieściłem w metodzie zastępującej metodę OnInit, odziedziczoną po klasie bazowej Control:

Kopiuj kod

protected override void OnInit(EventArgs e)

{

 Page.ClientScript.RegisterClientScriptResource(this.GetType(),

 "SlSphere.Silverlight.js");

 Page.ClientScript.RegisterClientScriptResource(this.GetType(),

 "SlSphere.SilverlightSphere.js");

 base.OnInit(e);

}

Ostatnim i najważniejszym elementem kontrolki jest implementacja wirtualnej metody Render. Metoda ta musi realizować dwa zadania — renderować znacznik <div>, będący elementem‑gospodarzem dla wtyczki Silverlight, i renderować osadzony skrypt tworzący wtyczkę Silverlight.

Referencję do pliku XAML można wygenerować, wywołując metodę GetWebResourceUrl klasy ClientScriptManager. Daje nam to gwarancję, że kontrolka Silverlight będzie inicjowana z wykorzystaniem pliku XAML osadzonego w podzespole jako zasób. Na koniec musimy określić unikalny identyfikator wtyczki Silverlight. W tym wypadku jest on generowany poprzez dodanie do identyfikatora kontrolki (this.ClientID) końcówki „\_ctrl” (patrz **ilustracja 5.**).

 Ilustracja 5. Implementacja wirtualnej metody Render

Kopiuj kod

Const string \_silverlightCreateScript = @"Silverlight.createObject(

 '{0}, document.getElementById('{1}'), '{2},

 {{ width:'300', height:'300', version:'1.0' }},

 {{ onError:null, onLoad:null }},

 null);";

protected override void Render(HtmlTextWriter writer)

{

 string script = string.Format(\_silverlightCreateScript,

 Page.ClientScript.GetWebResourceUrl(GetType(),

 "SlSphere.Sphere.xaml"),

 this.ClientID, this.ClientID + "\_ctrl");

 writer.AddAttribute(HtmlTextWriterAttribute.Id,

 this.ClientID);

 writer.RenderBeginTag(HtmlTextWriterTag.Div);

 writer.AddAttribute(HtmlTextWriterAttribute.Type,

 "text/javascript");

 writer.RenderBeginTag(HtmlTextWriterTag.Script);

 writer.Write(script);

 writer.RenderEndTag();

 writer.RenderEndTag();

 base.Render(writer);

}

W tej chwili mamy gotową kontrolkę serwerową (z plikami JavaScript i XAML osadzonymi jako zasoby), zawierającą wszystkie elementy niezbędne do osadzenia treści Silverlight w stronie ASP.NET. Instancję tej kontrolki na stronie można utworzyć za pomocą następującej deklaracji:

Kopiuj kod

<%@ Register Assembly="SlSphere" Namespace="MsdnMagazine"

 TagPrefix="csc" %>

...

<csc:SilverlightSphere ID="\_silverlightSphere" runat="server" />

Funkcjonalność udostępniana przez kontrolkę jest rewelacyjna — w dowolną aplikację ASP.NET można z łatwością wbudować powiększającą się i zmniejszającą kulę, wyświetlaną z wykorzystaniem Silverlight, nie martwiąc się o szczegóły związane z osadzaniem treści Silverlight.

Oczywiście można budować bardziej interesujące i złożone kontrolki, wyświetlające skomplikowane pliki XAML. Jednak gdy zaczniemy myśleć, do czego wykorzystać takie kontrolki, szybko staniemy przed problemem, jak przekazywać do XAML wartości właściwości kontrolki serwerowej. Chciałbym na przykład wyposażyć moją kontrolkę we właściwość Title i przekazywać jej wartość do właściwości Text elementu titleText w pliku XAML. Obecna implementacja nie oferuje żadnej prostej metody osiągnięcia takiego celu.

Inną wadą obecnej implementacji jest fakt, że plik XAML — będąc osadzonym zasobem — ma ustaloną zawartość. Pozwala to na wygodne wdrażanie kontrolki, jednakże jedną z najważniejszych zalet Silverlight i XAML miała być separacja definicji zachowań od definicji wyglądu, pozwalająca na wymienianie plików XAML. Mógłbym wyposażyć kontrolkę we właściwość XamlUrl, pozwalającą na zastąpienie wbudowanej treści XAML. Rozwiązanie to byłoby łatwe w implementacji, jednak mogłoby spowodować problemy z dopasowywaniem nazw procedur obsługi zdarzeń w kodzie XAML i JavaScript — wszyscy projektanci musieliby mieć tę samą, najnowszą wersję XAML.

# ASP.NET AJAX i IScriptControl

Kluczem do rozwiązania problemu jest znalezienie sposobu wprowadzenia właściwości dostępnych po stronie serwera do skryptu uruchamianego po stronie klienta. Na szczęście, funkcjonalność wiązania właściwości kontrolek serwerowych z elementami wyświetlanymi po stronie klienta udostępniana jest przez rozszerzenia ASP.NET AJAX za pośrednictwem interfejsu IScriptControl, zdefiniowanego w podzespole System.Web.Extensions. Interfejs ten definiuje dwie metody — GetScriptReferences i GetScriptDescriptors. Implementacje tych metod w powiązaniu z kontrolką ScriptManager strony umożliwiają generowanie referencji do plików JavaScript i deklarowanie odpowiednich instancji klas JavaScript:

Kopiuj kod

public interface IScriptControl

{

 IEnumerable<ScriptDescriptor> GetScriptDescriptors();

 IEnumerable<ScriptReference> GetScriptReferences();

}

Deskryptor można traktować jako działający po stronie klienta substytut kontrolki serwerowej, którego właściwości są inicjalizowane wartościami właściwości tej kontrolki. Budowane jest w ten sposób połączenie pomiędzy właściwościami kontrolki serwerowej a właściwościami po stronie klienta, dostępnymi dla Silverlight i procedur obsługi zdarzeń JavaScript. Deskryptory stanowią także mechanizm izolowania wszystkich widocznych po stronie klienta elementów kontrolki w przestrzeni nazw lub klasie (na tyle, na ile biblioteki klienckie ASP.NET AJAX definiują przestrzenie nazw i klasy w JavaScript).

Tworzenie kontrolki ASP.NET AJAX z osadzoną treścią Silverlight rozpocząłem od utworzenia definicji klasy działającej po stronie klienta, zawierającej metody i właściwości służące do interakcji z kontrolką Silverlight. Potrzebne nam będą właściwości Title i XamlUrl. Utworzyłem klasę MsdnMagazine.SilverlightAjaxSphere i zdefiniowałem w niej te właściwości:

Kopiuj kod

// File: SilverlightAjaxSphere.js

Type.registerNamespace('MsdnMagazine');

MsdnMagazine.SilverlightAjaxSphere = function(element) {

 MsdnMagazine.SilverlightAjaxSphere.initializeBase(

 this, [element]);

 this.\_title = null;

 this.\_xamlUrl = null;

}

Teraz muszę przygotować metody dla prototypu klasy, począwszy od metody initialize (patrz **ilustracja 6.**). Zamiast osadzać skrypt wewnątrz elementu <div>, tak jak w pierwszej implementacji kontrolki, tym razem tworzę wtyczkę Silverlight wewnątrz metody initialize klasy działającej po stronie klienta. Należy zauważyć, że jest to możliwe pod warunkiem, że zdefiniowana w klasie właściwość \_xamlUrl zostanie przed wywołaniem metody initialize zainicjowana wartością pobraną z serwera. Identyfikator elementu <div> także można uzyskać za pośrednictwem kontrolki serwerowej z wykorzystaniem kontrolki get\_id odziedziczonej po klasie bazowej Sys.UI.Control.

 Ilustracja 6. Treść metod prototypu klasy

Kopiuj kod

// plik: SilverlightAjaxSphere.js

MsdnMagazine.SilverlightAjaxSphere.prototype = {

 initialize : function() {

 MsdnMagazine.SilverlightAjaxSphere.callBaseMethod(

 this, 'initialize');

 var hostId = this.get\_id() + 'Host';

 Silverlight.createObject(this.\_xamlUrl,

 $get(this.get\_id()), hostId,

 { width:'300', height:'300', version:'1.0' },

 { onError:null,

 onLoad:Function.createDelegate(this, this.\_onXamlLoaded) },

 null);

 },

 dispose : function() {

 MsdnMagazine.SilverlightAjaxSphere.callBaseMethod(this,

 'dispose');

 },

 // po wywołaniu kontrolka Silverlight jest inicjowana

 // i elementy Xaml stają się dostępne.

 \_onXamlLoaded : function(root) {

 var root = $get(this.get\_id() + 'Host');

 var sphere = root.content.findName('ellipse');

 if (sphere) {

 sphere.addEventListener('MouseLeftButtonDown',

 Function.createDelegate(this,

 this.\_onSphereButtonDown));

 }

 else

 throw Error.invalidOperation(

 "You must have an ellipse element.");

 var title = root.content.findName('titleText');

 if (title)

 title.Text = this.\_title;

 },

 \_onSphereButtonDown : function(sender, args)

 {

 // implementacja taka sama, jak poprzednio

 },

 // procedury dostępu do właściwości

 //

 get\_title : function() {

 return this.\_title;

 },

 set\_title : function(value) {

 if (this.\_title !== value)

 this.\_title = value;

 },

 get\_xamlUrl : function() {

 return this.\_xamlUrl;

 },

 set\_xamlUrl : function(value) {

 if (this.\_xamlUrl !== value)

 this.\_xamlUrl = value;

 }

}

MsdnMagazine.SilverlightAjaxSphere.registerClass(

 'MsdnMagazine.SilverlightAjaxSphere', Sys.UI.Control);

if (typeof(Sys) !== 'undefined')

 Sys.Application.notifyScriptLoaded();

Gdy metoda initialize niniejszej klasy jest wywoływana, kontrolka Silverlight jest właśnie tworzona, a kod XAML nie został jeszcze odczytany. Dlatego korzystam z procedury obsługi zdarzeń onLoad, definiowanej z wykorzystaniem metody createObject obiektu Sys.Silverlight, w celu ustawienia funkcji zwrotnej, która zostanie wywołana po załadowaniu kodu XAML. Pozwala mi to podłączyć procedury obsługi zdarzeń i zainicjować właściwości poszczególnych elementów XAML. Do procedury obsługi tego zdarzenia, którą nazwałem \_onXamlLoaded, jest przekazywana referencja do elementu‑korzenia Silverlight, która pozwala na pobieranie nazwanych elementów z kodu XAML.

Odwołując się do nazwanych elementów XAML, należy zachować szczególną ostrożność i przed modyfikacją ich zawartości lub przed przypisaniem procedury obsługi zdarzeń sprawdzać, czy elementy te istnieją. Jeśli element nie istnieje i nie jest niezbędny do poprawnej pracy kontrolki (tak jak element titleText w mojej kontrolce), można po prostu pominąć ich inicjalizację. Dzięki temu projektant może zdecydować się na usunięcie określonych elementów z kodu XAML, a kontrolka nadal będzie działać poprawnie.

Pozostała część implementacji klasy definiuje procedury dostępu do właściwości title i xamlUrl oraz implementuje funkcję \_onSphereButtonDown, inicjującą animację powiększenia i zmniejszenia kuli. Implementacja ta jest identyczna, jak w pierwszej wersji kontrolki, jednak tym razem jest ona elementem klasy.

Końcowa część skryptu uruchamianego po stronie klienta służy rejestracji nowej klasy ze wskazaniem klasy bazowej Sys.UI.Control.

Teraz musimy zbudować kontrolkę serwerową komunikującą się z przygotowaną przed chwilą klasą za pośrednictwem interfejsu IScriptControl (patrz **ilustracja 7.**). Analogicznie jak klasa pracująca po stronie klienta, kontrolka serwerowa posiada dwie właściwości — Title i XamlUrl (właściwości te uzupełniane są przez właściwość ViewState, ponieważ wszystkie kontrolki serwerowe muszą posiadać taką właściwość). Właściwość XamlUrl zainicjowałem referencją do osadzonego pliku XAML w taki sam sposób, jak poprzednio, korzystając z metody GetWebResourceUrl. Ustawiając tę właściwość w metodzie zastępującej OnInit mam pewność, że wartością domyślną tej właściwości będzie adres osadzonego pliku XAML, ale gdy klient zdecyduje się przygotować własną wersję pliku, nowa wersja będzie traktowana priorytetowo, ponieważ stan kontrolki jest ładowany zawsze po jej inicjalizacji.

 Ilustracja 7. Kontrolka serwerowa

Kopiuj kod

// plik: SilverlightAjaxSphere.cs

//

using System;

using System.Collections.Generic;

using System.Text;

using System.Web;

using System.Web.UI;

[assembly: WebResource("SlAjaxSphere.SilverlightAjaxSphere.js",

 "text/javascript")]

[assembly: WebResource("SlAjaxSphere.Silverlight.js",

 "text/javascript")]

[assembly: WebResource("SlAjaxSphere.Sphere.xaml", "text/xml")]

namespace MsdnMagazine

{

 public class SilverlightAjaxSphere : Control, IScriptControl

 {

 public string Title

 {

 get { return (string)(ViewState["title"] ?? "[title]"); }

 set { ViewState["title"] = value; }

 }

 public string XamlUrl

 {

 get { return (string)(ViewState["xamlurl"] ??

 string.Empty); }

 set { ViewState["xamlurl"] = value; }

 }

 protected override void OnInit(EventArgs e)

 {

 // Domyślnie XamlUrl wskazuje na osadzony plik zasobu.

 // Jeśli właściwość zostanie jawnie osadzona w kodzie aplikacji,

 // wartość domyślna zostanie zastąpiona.

 XamlUrl = Page.ClientScript.GetWebResourceUrl(GetType(),

 "SlAjaxSphere.Sphere.xaml");

 base.OnInit(e);

 }

 public IEnumerable<ScriptReference> GetScriptReferences()

 {

 ScriptReference sr1 = new ScriptReference(

 " SlAjaxSphere.Silverlight.js",

 GetType().Assembly.FullName);

 ScriptReference sr2 = new ScriptReference(

 " SlAjaxSphere.SilverlightAjaxSphere.js",

 GetType().Assembly.FullName);

 return new ScriptReference[] { sr1, sr2 };

 }

 public IEnumerable<ScriptDescriptor> GetScriptDescriptors()

 {

 ScriptControlDescriptor scd = new ScriptControlDescriptor(

 "MsdnMagazine.SilverlightAjaxSphere",

 this.ClientID);

 scd.AddProperty("title", this.Title);

 scd.AddProperty("xamlUrl", this.XamlUrl);

 return new ScriptDescriptor[] { scd };

 }

 protected override void OnPreRender(EventArgs e)

 {

 if (!DesignMode)

 {

 ScriptManager sm = ScriptManager.GetCurrent(Page);

 if (sm == null)

 throw new HttpException(

 "A ScriptManager control must exist on the current page.");

 sm.RegisterScriptControl(this);

 sm.RegisterScriptDescriptors(this);

 }

 base.OnPreRender(e);

 }

 protected override void Render(HtmlTextWriter writer)

 {

 writer.AddAttribute(HtmlTextWriterAttribute.Id,

 this.ClientID);

 writer.RenderBeginTag(HtmlTextWriterTag.Div);

 writer.RenderEndTag();

 base.Render(writer);

 }

 }

}

Kolejne dwie metody — GetScriptDescriptors i GetScriptReferences — zawierają implementację interfejsu IScriptControl. Ponieważ pliki JavaScript są osadzone w podzespole jako pliki zasobów, korzystam z przeciążonego konstruktora klasy ScriptReference, przyjmującego jako parametry nazwę zasobu i nazwę podzespołu. Powoduje to wygenerowanie referencji do osadzonego zasobu skryptu w sposób podobny, jak realizowany przez metodę RegisterClientScriptResource w poprzedniej implementacji kontrolki. Metoda GetScriptDescriptors pozwala na utworzenie instancji klasy MsdnMagazine.SilverlightAjaxSphere, zainicjowanej wartościami właściwości pobranymi z kontrolki serwerowej.

Konstruktor klasy ScriptControlDescriptor jako parametry przyjmuje nazwę klasy działającej po stronie klienta oraz identyfikator elementu związanego z kontrolką Silverlight. Po utworzeniu instancji klasy działającej po stronie klienta, inicjuję wartość poszczególnych właściwości tej klasy, korzystając z udostępnianej przez nią metody AddProperty. Przekazując bieżące wartości właściwości kontrolki mam pewność, że wartości właściwości klasy utworzonej po stronie klienta będą dokładnie odpowiadały wartościom zdefiniowanym w kontrolce.

Muszę się jeszcze zająć metodą zastępującą OnPreRender, w której za pośrednictwem obiektu ScriptManager bieżącej strony (i wyrzucając wyjątek, jeżeli nie ma takiego obiektu) rejestruję kontrolkę jako kontrolkę korzystającą ze skryptów i deskryptorów skryptów. Powoduje to wywołanie przez obiekt ScriptManager implementowanych przez kontrolkę dwóch metod interfejsu IScriptControl w celu pobrania referencji do skryptów i utworzenia początkowej postaci klasy działającej po stronie klienta. Metoda Render pozostaje identyczna, jak w poprzedniej implementacji kontrolki, a jej działanie polega na utworzeniu elementu <div>, służącego jako element‑gospodarz dla wtyczki Silverlight.

Mamy już kompletną kontrolkę serwerową z osadzoną treścią Silverlight. W tej implementacji kontrolka udostępnia właściwości pozwalające na inicjowanie elementów XAML (lub, w razie potrzeby, nawet zastąpienie całego kodu XAML). Aby na stronie ASP.NET umieścić instancję nowej kontrolki, w której element title zawiera określony tekst, należy użyć następującej deklaracji:

Kopiuj kod

<%@ Register Assembly="SlAjaxSphere" Namespace="MsdnMagazine"

 TagPrefix="sas" %>

...

<sas:SilverlightSphere ID="\_silverlightSphere" runat="server"

 Title="My custom title!" />

# Kontrolki asp:Xaml i asp:Media

W zestawie dodatków ASP.NET Futures z maja 2007 roku wprowadzono dwie nowe kontrolki ASP.NET AJAX — asp:Xaml i asp:Media (przy czym kontrolka asp:Media jest pochodną kontrolki asp:XAML). Obie kontrolki zapewniają własne sposoby osadzania treści Silverlight i obie zostały utworzone w bardzo podobny sposób, jak przykładowa kontrolka przedstawiona w poprzedniej sekcji. Prawdę mówiąc, kontrolki te zainspirowały mnie do utworzenia mojej kontrolki.

Kontrolka asp:Xaml pozwala na przypisanie pliku XAML, zawiera właściwości służące do tworzenia wtyczki Silverlight i samodzielnie obsługuje cały proces tworzenia wtyczki i ładowania pliku XAML. Aby z wykorzystaniem tej kontrolki wyświetlić nasz plik Sphere.xaml, należy użyć następującej deklaracji:

Kopiuj kod

<asp:Xaml runat="server" ID="\_sphereXaml" Windowless="true"

 Width="300px" Height="300px"

 XamlUrl="~/Sphere.xaml" />

Kontrolka asp:Media pozwala na łatwe osadzanie treści multimedialnych (wideo i audio) w stronach ASP.NET i jest dobrym przykładem możliwości, jakie daje tworzenie kontrolek serwerowych ASP.NET, zawierających treść Silverlight. Jedną z najciekawszych cech kontrolki Media jest to, że zawiera osiem różnych karnacji (skin), które są plikami XAML. Do prezentacji treści multimedialnych można wykorzystać jedną z gotowych karnacji lub zaprojektować własną. Na **ilustracji 8.** przedstawiono kontrolkę Media wyświetlającą wideo z wykorzystaniem karnacji Expression.



Ilustracja 8. **Kontrolka Media wyświetlająca wideo**

Kopiuj kod

<asp:Media runat="server" ID="\_butterflyVideo"

 MediaSkin="Expression" MediaUrl="~/Butterfly.wmv" />

Inną metodą utworzenia własnej kontrolki serwerowej Silverlight jest wykorzystanie kontrolki asp:Xaml jako kontrolki bazowej i wzbogacenie jej o właściwości i metody w sposób podobny, jak w przypadku drugiej implementacji naszej przykładowej kontrolki. W kodzie dostępnym do pobrania wraz z tym artykułem zamieściłem kolejną wersję kontrolki SilverlightSphere, utworzoną właśnie w taki sposób — poprzez dziedziczenie po kontrolce asp:Xaml.

Wszelkie pytania i komentarze proszę przesyłać na adres xtrmasp@microsoft.com.

**Fritz Onion** jest współzałożycielem firmy Pluralsight, oferującej szkolenia w zakresie Microsoft .NET, w której prowadzi kursy z zakresu programowania aplikacji internetowych. Fritz jest autorem książek: Essential ASP.NET (Addison Wesley, 2003) i Essential ASP.NET 2.0 (Addison Wesley, 2006). Można się z nim skontaktować za pośrednictwem strony [pluralsight.com/fritz](http://pluralsight.com/fritz).