<http://msdn.microsoft.com/en-us/magazine/cc721608.aspx>

**Silverlight**

**Tworzenie animacji z wykorzystaniem XAML i Expression Blend**

Laurence Moroney

|  |  |
| --- | --- |
| **Zagadnienia, których dotyczy artykuł:*** Transformacje proste i złożone
* Stosowanie wyzwalaczy i zdarzeń
* Animacje liniowe i oparte na dyskretnych klatkach kluczowych
* Tworzenie animacji z wykorzystaniem Expression Blend
 | **Artykuł dotyczy następujących technologii:**Silverlight 2, Expression Blend |

Artykuł oparto na przedpremierowej wersji Silverlight 2. Przedstawione tu informacje mogą ulec zmianie. Artykuł został zaadaptowany z książki „*Introducing Microsoft Silverlight 2”, wydanie drugie,* Laurence Moroney, Microsoft Press, 2008.

 Spis treści

[Transformacje](#_Toc206673158)

[Obracanie obiektów z wykorzystaniem RotateTransform](#_Toc206673159)

[Skalowanie obiektów z wykorzystaniem właściwości ScaleTransform](#_Toc206673160)

[Przesuwanie obiektu z wykorzystaniem właściwości TranslateTransform](#_Toc206673161)

[Pochylenie obiektu z wykorzystaniem właściwości SkewTransform](#_Toc206673162)

[Definiowanie transformacji za pomocą MatrixTransform](#_Toc206673163)

[Łączenie transformacji](#_Toc206673164)

[Animacje](#_Toc206673165)

[Stosowanie obiektów Trigger i EventTrigger](#_Toc206673166)

[Stosowanie obiektów BeginStoryboard i Storyboard](#_Toc206673167)

[Definiowanie parametrów animacji](#_Toc206673168)

[Definiowanie obiektu docelowego animacji](#_Toc206673169)

[Ustawianie parametrów właściwości](#_Toc206673170)

[Ustawianie właściwości RepeatBehavior](#_Toc206673171)

[Animowanie wartości z wykorzystaniem obiektu DoubleAnimation](#_Toc206673172)

[Animowanie koloru z wykorzystaniem obiektu ColorAnimation](#_Toc206673173)

[Animowanie punktu z wykorzystaniem obiektu PointAnimation](#_Toc206673174)

[Stosowanie klatek kluczowych](#_Toc206673175)

[Stosowanie liniowych klatek kluczowych](#_Toc206673176)

[Stosowanie dyskretnych klatek kluczowych](#_Toc206673177)

[Stosowanie klatek kluczowych typu spline](#_Toc206673178)

[Animacja a Expression Blend](#_Toc206673179)

[Podsumowanie](#_Toc206673180)

**Jedną z najciekawszych cech XAML** jest to, że składnia XAML pozwala nie tylko na deklarowanie obiektów, ale także na definiowanie transformacji przekształcających te obiekty. By móc obracać, przesuwać czy pochylać obiekty, nie trzeba być programistą. XAML powala także opisać sposób animacji obiektu, przy czym animacja definiowana jest jako zmienianie właściwości obiektu w czasie. Najpierw zajmę się omówieniem transformacji. Później przejdę do linii czasu pozwalających na animowanie treści Silverlight™.

Transformacje

W grafice komputerowej transformacja jest odwzorowaniem przekształcającym jeden zbiór punktów w inny. Transformacje zwykle opisuje się za pomocą macierzy transformacji — tworu matematycznego pozwalającego na przekształcanie zbiorów punktów za pośrednictwem prostych obliczeń matematycznych. Składnia języka XAML w Silverlight zastępuje macierze czterema podstawowymi transformacjami — obrotem, jednokładnością (skalowaniem), pochyleniem i translacją (przesunięciem). W Silverlight XAML wprowadzono dodatkowy, specjalny typ transformacji, pozwalający na zdefiniowanie i zaimplementowanie własnej macierzy, którą można następnie wykorzystać do składania transformacji ze sobą.

Transformacje realizowane są za pośrednictwem właściwości transformacji. Dostępne jest wiele różnych typów właściwości transformacji, które można zastosować do różnych typów obiektów.

Dlatego, stosując np. typ Brush, można zdefiniować transformację na różne sposoby. Chcąc zmodyfikować zawartość obiektu Brush (na przykład w celu obrócenia pliku graficznego, zanim zostanie wykorzystany w obiekcie ImageBrush), należy użyć właściwości Brush.Transform. Natomiast właściwość Brush.RelativeTransform pozwala na realizację transformacji z wykorzystaniem wartości względnych (może to być potrzebne na przykład do wypełnienia różnych obszarów o różnych rozmiarach z wykorzystaniem tego samego obiektu Brush).

Stosując typ Geometry, prostą transformację można zrealizować korzystając z właściwości Geometry.Trasnform. Warto zauważyć, że ten typ nie obsługuje transformacji względnych.

Natomiast transformacje dla elementów interfejsu użytkownika można definiować za pomocą właściwości RenderTransform. Chcąc na przykład zrealizować transformację elipsy, należy użyć właściwości Ellipse.RenderTransform.

Obracanie obiektów z wykorzystaniem RotateTransform

Element RotateTransform pozwala na obracanie elementów o określony kąt wokół określonego punktu obrotu. Kąt obrotu podawany jest za pomocą właściwości Angle, określającej liczbę stopni, o którą należy obrócić element. By zrozumieć zasadę działania tej transformacji, wyobraźmy sobie poziomy wektor skierowany w prawo. Jest to pozycja 0 stopni. Obrót realizowany jest zgodnie z kierunkiem ruchu wskazówek zegara — wynikiem obrotu o 90 stopni będzie pionowy wektor skierowany w dół.

Punkt obrotu określany jest za pomocą właściwości CenterX i CenterY. Domyślną wartością tych właściwości jest 0, 0, więc domyślny punkt obrotu znajduje się w lewym górnym rogu elementu-pojemnika (container).

W poniższym przykładzie element TextBlock jest obracany o 45 stopni z wykorzystaniem właściwości RenderTransform, zawierającej transformację RotateTransform.

Skopiuj kod

<TextBlock Width="320" Height="40"

 Text="This is the text to rotate" TextWrapping="Wrap">

 <TextBlock.RenderTransform>

 <RotateTransform Angle="45" />

 </TextBlock.RenderTransform>

</TextBlock>

Jak widać na **ilustracji 1**, tekst jest obracany dookoła punktu o współrzędnych (0, 0) — w lewym górnym rogu ekranu.



Ilustracja 1. **Użycie właściwości RotateTransform** (aby powiększyć ilustrację, kliknij ją)

W poniższym przykładzie zastosowano atrybuty CenterX i CenterY w celu wykonania obrotu wokół innego punktu. W tym przypadku obrót wykonywany jest wokół punktu o współrzędnych (100, 200).

Skopiuj kod

<TextBlock Width="320" Height="40"

 Text="This is the text to rotate" TextWrapping="Wrap" >

 <TextBlock.RenderTransform>

 <RotateTransform Angle="45" CenterX="100" CenterY="200" />

 </TextBlock.RenderTransform>

</TextBlock>

Skalowanie obiektów z wykorzystaniem właściwości ScaleTransform

Właściwość ScaleTransform pozwala na zmianę rozmiarów obiektu wzdłuż osi poziomej, pionowej lub obu osi jednocześnie. Skalując obiekt, należy określić przynajmniej jedną oś, wzdłuż której obiekt ma być skalowany, oraz współczynnik skalowania wzdłuż tej osi.

Do skalowania obiektu wzdłuż osi poziomej (osi X) służy właściwość ScaleX, a do skalowania wzdłuż osi pionowej (osi Y) — właściwość ScaleY. Wartość obu właściwości jest typu double i określa wartość, przez którą ma zostać pomnożony obecny rozmiar obiektu wzdłuż danej osi. Dlatego użycie wartości większej niż 1 spowoduje powiększenie obiektu — na przykład użycie właściwości ScaleX o wartości 2 spowoduje, że rozmiar poziomy obiektu będzie dwa razy większy. Wartości mniejsze niż 1, ale większe niż 0 powodują zmniejszanie obiektu. Na przykład użycie wartości 0,5 spowoduje zmniejszenie rozmiaru obiektu o połowę.

Poniższy kod XAML powoduje wyświetlenie czerwonego prostokąta o rozmiarach 96 na 88 pikseli.

Skopiuj kod

<Rectangle Fill="#FFFF0404" Stroke="#FF000000"

 Width="96" Height="88"

 Canvas.Left="112" Canvas.Top="72" />

Na **ilustracji 2.** przedstawiono wygląd tego obiektu po wyświetleniu w Silverlight.



Ilustracja 2. **Wyświetlenie prostokąta** (aby powiększyć ilustrację, kliknij ją)

Teraz przeskalujemy ten obiekt używając właściwości RenderTransform i transformacji ScaleTransform.

Skopiuj kod

<Rectangle Fill="#FFFF0404" Stroke="#FF000000"

 Width="96" Height="88" Canvas.Left="112" Canvas.Top="72">

 <Rectangle.RenderTransform>

 <ScaleTransform ScaleX="2" />

 </Rectangle.RenderTransform>

</Rectangle>

Gdy wyświetlimy zmodyfikowaną stronę, zobaczymy, że zastosowanie transformacji ScaleTransform spowodowało wydłużenie prostokąta w poziomie w prawo. Jest tak, ponieważ nie określiliśmy punktu centralnego operacji skalowania. Punkt ten można określić za pomocą właściwości CenterX w przypadku skalowania poziomego i CenterY dla skalowania pionowego. Właściwości te pozwalają na określenie współrzędnych punktu centralnego transformacji skalowania. Współrzędne te określane są względem lewego górnego roku prostokąta. Domyślną wartością tych właściwości jest 0, dlatego skalowanie poziome powoduje wydłużanie obiektów w prawo, a skalowanie pionowe — w dół.

Jeśli właściwość CenterX będzie miała wartość dodatnią (na przykład 50), skalowanie będzie realizowane względem punktu znajdującego się 50 pikseli w prawo od lewej krawędzi prostokąta. Prostokąt będzie wyglądał tak, jakby został przesunięty o pewną odległość w lewo względem prostokąta, w którym właściwość CenterX nie została ustawiona (odległość ta zależy od wielkości prostokąta i współczynnika skalowania). Jest to spowodowane przesunięciem punktu centralnego — lewa część prostokąta jest wydłużana w lewo, a prawa w prawo. Użycie właściwości ScaleY i CenterY daje analogiczny efekt.

Skopiuj kod

<Rectangle Fill="#FFFF0404" Stroke="#FF000000"

 Width="96" Height="88"

 Canvas.Left="80" Canvas.Top="80">

 <Rectangle.RenderTransform>

 <ScaleTransform ScaleX="2"

 CenterX="50"/>

 </Rectangle.RenderTransform>

</Rectangle>

Przesuwanie obiektu z wykorzystaniem właściwości TranslateTransform

Translacja jest transformacją pozwalającą przesunąć obiekt na płaszczyźnie dwuwymiarowej z jednego miejsca na inne. Definiowana jest poprzez wektor określający przesunięcie obiektu wzdłuż osi X i Y. Wektor ten wyznaczany jest za pomocą właściwości X i Y transformacji. Aby przesunąć obiekt o dwie jednostki w poziomie (co oznacza, że zostanie on przesunięty w prawo), należy właściwości X przypisać wartość 2. Aby przesunąć obiekt w lewo, należy użyć wartości ujemnej — na przykład -2. Analogicznie, by przesunąć obiekt w pionie, należy użyć właściwości Y. Wartości dodatnie powodują przesuwanie obiektu w dół, natomiast ujemne — w górę.

Oto przykład transformacji translacji przesuwającej nasz czerwony prostokąt w górę i w lewo o zadane wartości x i y. Wartości te tworzą wektor określający transformację.

Skopiuj kod

<Rectangle Fill="#FFFF0404"

 Stroke="#FF000000"

 Width="96" Height="88"

 Canvas.Left="80" Canvas.Top="80">

 <Rectangle.RenderTransform>

 <TranslateTransform X="-50" Y="-50"/>

 </Rectangle.RenderTransform>

</Rectangle>

Efekty przedstawiono na **ilustracji 3**. Prostokąt został przesunięty w górę i w lewo względem swojej pozycji początkowej. Można to sprawdzić, porównując tę ilustrację z **ilustracją 2**.



Ilustracja 3. **Zastosowanie właściwości TranslateTransform** (aby powiększyć ilustrację, kliknij ją)

Pochylenie obiektu z wykorzystaniem właściwości SkewTransform

Pochylenie obiektu polega na progresywnej modyfikacji tego obiektu wzdłuż określonej osi. Prostokąt lub kwadrat poddane takiemu przekształceniu stają się równoległobokami. Transformację tę wykorzystuje się często w celu uzyskania efektu głębi na płaszczyźnie dwuwymiarowej.

Pochylenie o określony kąt można zastosować zarówno wzdłuż osi X, jak i Y. Analogicznie jak w poprzednio omówionych transformacjach, można także zdefiniować punkt centralny. Możliwe jest również jednoczesne zrealizowanie pochylenia względem osi X i Y. W poniższym przykładzie nasz prostokąt zostaje pochylony wzdłuż osi X o kąt 45 stopni.

Skopiuj kod

<Rectangle Fill="#FFFF0404" Stroke="#FF000000"

 Width="96" Height="88"

 Canvas.Left="80" Canvas.Top="80">

 <Rectangle.RenderTransform>

 <SkewTransform AngleX="45"/>

 </Rectangle.RenderTransform>

 </Rectangle>

Efekty można zobaczyć na **ilustracji 4**.



Ilustracja 4. **Pochylenie prostokąta z wykorzystaniem transformacji SkewTransform** (aby powiększyć ilustrację, kliknij ją)

Transformacja pochylenia jest przydatna w symulowaniu efektów trójwymiarowych w grafice. Na przykład stosując transformację SkewTransform do trzech sąsiadujących ze sobą prostokątów (dwa pochylone wzdłuż osi X i jeden wzdłuż osi Y), można uzyskać złudzenie perspektywy trójwymiarowej (patrz **ilustracja 5**).



Ilustracja 5. **Symulacja perspektywy za pomocą trzech pochylonych prostokątów** (aby powiększyć ilustrację, kliknij ją)

Definiowanie transformacji za pomocą MatrixTransform

Wszystkie transformacje w gruncie rzeczy realizowane są poprzez przemnożenie współrzędnych poszczególnych punktów obiektu przez macierz transformacji. Każda z omówionych do tej pory transformacji jest dobrze znaną transformacją posiadającą określoną definicję. Obliczenia macierzowe i sposób implementacji transformacji leżą poza zakresem tematycznym tego artykułu, jednak chcąc omówić całą składnię transformacji, muszę omówić sposób definiowania macierzy w Silverlight XAML.

Warto zwrócić uwagę, że macierze transformacji zawsze są macierzami afinicznymi, co oznacza, że ostatnia kolumna macierzy transformacji MatrixTransform zawsze ma postać (0 0 1), więc wystarczy ustawić wartości tylko w pierwszych dwóch kolumnach. Można tego dokonać za pomocą właściwości Matrix, która przyjmuje łańcuch znaków zawierający rozdzielone spacjami wartości komórek dwóch pierwszych kolumn macierzy.

Skopiuj kod

<Rectangle Fill="#FFFF0404"

 Stroke="#FF000000"

 Width="96" Height="88"

 Canvas.Left="80" Canvas.Top="80">

 <Rectangle.RenderTransform>

 <MatrixTransform Matrix="1 0 1 2 0 1"/>

 </Rectangle.RenderTransform>

 </Rectangle>

Efektem użycia transformacji, zdefiniowanej za pomocą przedstawionej w przykładzie macierzy, jest jednoczesne rozciągnięcie i pochylenie prostokąta.

Łączenie transformacji

Jak można było przekonać się w poprzednim przykładzie, możliwe jest tworzenie transformacji złożonych w oparciu o macierz przekształcenia afinicznego, określaną za pośrednictwem typu MatrixTransform. Nie każdy jednak jest ekspertem w obliczeniach macierzowych, dlatego przewidziano inną metodę łączenia transformacji. Można w tym celu wykorzystać element TransformGroup. Pozwala on na zdefiniowanie wielu transformacji, a następnie jednoczesne zastosowanie ich do obiektu.

Skopiuj kod

<Rectangle Fill="#FFFF0404" Stroke="#FF000000"

 Width="96" Height="88"

 Canvas.Left="80" Canvas.Top="80">

 <Rectangle.RenderTransform>

 <TransformGroup>

 <ScaleTransform ScaleX="1.2" ScaleY="1.2" />

 <SkewTransform AngleX="30" />

 <RotateTransform Angle="45" />

 </TransformGroup>

 </Rectangle.RenderTransform>

</Rectangle>

W przykładzie połączono transformację ScaleTransform, zwiększającą rozmiary obiektu o 20 procent wzdłuż obu osi, 30-stopniowe pochylenie obiektu wzdłuż osi X i obrót o 45 stopni.

Animacje

Słowo animacja oznacza dosłownie „ożywienie czegoś”. Dlatego dzięki animacji możemy tchnąć życie w nasze projekty, zmieniając właściwości — takie jak kolor, rozmiar, przezroczystość i inne — w czasie lub w odpowiedzi na działania użytkownika.

W XAML animacja obiektu polega na zmienianiu jego właściwości w czasie. Czas definiowany jest z wykorzystaniem linii czasu. Na przykład chcąc przesunąć obiekt przez cały ekran w ciągu pięciu sekund, należy zdefiniować pięciosekundową linię czasu, która animuje właściwość Canvas.Left od wartości 0 do wartości równej szerokości ekranu. W następnych sekcjach omówię każdy z dostępnych typów animacji oraz powiem, jak animować właściwości z wykorzystaniem klatek kluczowych.

Przed zajęciem się poszczególnymi typami animacji musimy zapoznać się z infrastrukturą umożliwiającą tworzenie animacji, składającą się z obiektów typu Trigger, EventTrigger i Storyboard. Dlatego — zanim omówię poszczególne typy animacji — zajmę się tymi podstawowymi pojęciami.

Stosowanie obiektów Trigger i EventTrigger

Animacje w Silverlight uruchamiane są w odpowiedzi na zdarzenie definiowane za pomocą wyzwalacza. Obecnie Silverlight obsługuje tylko jeden typ wyzwalacza — EventTrigger. Z każdą właściwością interfejsu użytkownika związana jest kolekcja Triggers, która pozwala na zdefiniowanie jednego lub kilku wyzwalaczy, czyli jednego lub kilku obiektów Trigger.

Tak więc pierwszym krokiem w definiowaniu animacji elementu jest zdefiniowanie kolekcji Triggers. Następnie do kolekcji tej należy dodać co najmniej jeden obiekt EventTrigger. W przypadku animacji na przykład prostokąta, pierwszy krok — zdefiniowanie kolekcji Triggers — wygląda następująco:

Skopiuj kod

<Rectangle x:Name="rect" Fill="Red"

 Canvas.Top="100" Canvas.Left="100"

 Width="100" Height="100">

 <Rectangle.Triggers>

 </Rectangle.Triggers>

</Rectangle>

Następnie należy zdefiniować obiekt EventTrigger, który dodamy do kolekcji. Do wskazania zdarzenia, w odpowiedzi na które wykona się animacja, użyjemy właściwości RoutedEvent. Właściwość RoutedEvent obsługuje wyłącznie zdarzenie Loaded.

Aby zaimplementować animację, która rozpocznie się w momencie załadowania prostokąta, EventTrigger należy zdefiniować w następujący sposób:

Skopiuj kod

<EventTrigger RoutedEvent="Rectangle.Loaded">

</EventTrigger>

Kod XAML uruchamiający animację wygląda następująco:

Skopiuj kod

<Rectangle x:Name="rect" Fill="Red" Canvas.Top="100"

 Canvas.Left="100" Width="100" Height="100">

 <Rectangle.Triggers>

 <EventTrigger RoutedEvent="Rectangle.Loaded">

 </EventTrigger>

 </Rectangle.Triggers>

</Rectangle>

Następnym krokiem jest zdefiniowanie animacji. Do definiowania animacji służą obiekty Storyboard.

Stosowanie obiektów BeginStoryboard i Storyboard

BeginStoryboard to działanie wyzwalające zawierające obiekt Storyboard. Natomiast obiekt Storyboard zawiera definicje animacji. Definiując animację, obiekty tego typu umieszcza się wewnątrz definicji EventTrigger. W przypadku naszego prostokąta obiekty te należy zastosować w następujący sposób:

Skopiuj kod

<Rectangle x:Name="rect" Fill="Red"

 Canvas.Top="100" Canvas.Left="100"

 Width="100" Height="100">

 <Rectangle.Triggers>

 <EventTrigger RoutedEvent="Rectangle.Loaded">

 <BeginStoryboard>

 <Storyboard>

 </Storyboard>

 </BeginStoryboard>

 </EventTrigger>

 </Rectangle.Triggers>

</Rectangle>

Definiowanie parametrów animacji

Przygotowaliśmy już wszystko, co jest potrzebne do uruchomienia animacji i możemy rozpocząć definiowanie animacji. Najprościej mówiąc, animacja definiuje zmianę właściwości w czasie. Animacji mogą podlegać trzy różne typy właściwości. Każdy z tych trzech typów animowany jest od wartości podanej w atrybucie From (lub — jeśli wartość ta nie została podana — od wartości bieżącej) do wartości określonej w atrybucie To. Można też użyć atrybutu By określającego, o ile dana wartość ma zostać zmieniona.

* **Typ Double** — właściwości tego typu mogą być animowane z użyciem DoubleAnimation lub DoubleAnimationUsingKeyFrames. Metodę tę można stosować do animowania właściwości, których wartość jest typu Double — czyli między innymi wymiarów (na przykład Canvas.Left) lub atrybutów wizualnych (na przykład Opacity).
* **Typ Point** — właściwości tego typu mogą być animowane z użyciem PointAnimation lub PointAnimationUsingKeyFrames. Metodę tę można stosować do animowania właściwości, których wartość jest typu Point — czyli na przykład segmentów linii lub krzywych definiowanych z użyciem punktów.
* **Typ Color** — właściwości tego typu mogą być animowane z użyciem ColorAnimation lub ColorAnimationUsingKeyFrames. Metodę tę można stosować do animowania właściwości określających kolor — na przykład kolor tła lub linii.

Definiowanie obiektu docelowego animacji

Do wskazania obiektu, którego ma dotyczyć animacja, służy właściwość Storyboard.TargetName typów animacji opisanych powyżej. We właściwości tej należy podać nazwę obiektu, zdefiniowaną dla tego obiektu za pomocą właściwości x:Name. Natomiast do wskazania właściwości, która będzie podlegała animacji, służy właściwość Storyboard.TargetProperty. Warto pamiętać, że nazwy złożone oraz nazwy właściwości dołączonych (na przykład Canvas.Left) należy umieszczać w nawiasach. Tak więc w celu zdefiniowania animacji typu Double, dotyczącej właściwości Canvas.Left prostokąta o nazwie rect, należy użyć następującego kodu XAML:

Skopiuj kod

<DoubleAnimation Storyboard.TargetName="rect"

 Storyboard.TargetProperty="(Canvas.Left)" />

Ustawianie parametrów właściwości

Do określenia, jak długo ma trwać przejście właściwości od jednej wartości do drugiej, służy właściwość Duration. Wartości tej właściwości podawane są w formacie GG:MM:SS. W przypadku animacji trwającej 5 sekund należy podać 00:00:05. Zapis ten można skrócić do 0:0:5.

Skopiuj kod

<DoubleAnimation Storyboard.TargetName="rect"

 Storyboard.TargetProperty="(Canvas.Left)" Duration="0:0:5" />

Jeśli nie chcemy, by animacja rozpoczęła się natychmiast, możemy ją opóźnić za pomocą właściwości BeginTime, której wartości podawane są w tym samym formacie.

Skopiuj kod

<DoubleAnimation Storyboard.TargetName="rect"

 Storyboard.TargetProperty="(Canvas.Left)" BeginTime="0:0:5" />

Czas animacji można modyfikować, mnożąc go przez współczynnik szybkości, definiowany za pomocą właściwości SpeedRatio. W poprzednim przykładzie czas animacji ustawiliśmy na 5 sekund. Zmieniając współczynnik szybkości, czyli ustawiając właściwość SpeedRatio na przykład na 2, możemy spowodować, że animacja będzie trwała 10 sekund. Ustawienie SpeedRatio na 0,2 spowoduje, że czas trwania animacji będzie wynosił 1 sekundę.

Skopiuj kod

<DoubleAnimation Storyboard.TargetName="rect"

 Storyboard.TargetProperty="(Canvas.Left)"

 SpeedRatio="2" Duration="0:0:5" />

W Silverlight istnieje możliwość odwrócenia zmian wprowadzonych przez animację. Na przykład jeżeli w określonym czasie animacja powoduje zmianę wartości typu Double z 0 na 500, włączenie funkcji autorewers spowoduje, że animacja wróci z 500 z powrotem do 0.

Uwaga — jeżeli animacja trwa 5 sekund i właściwość AutoReverse ma wartość true, cała animacja będzie trwała 10 sekund. Sposób włączenia funkcji autorewers przedstawiono w poniższym przykładzie.

Skopiuj kod

<DoubleAnimation Storyboard.TargetName="rect"

 Storyboard.TargetProperty="(Canvas.Left)"

 AutoReverse="True"

 Duration="0:0:5" />

Ustawianie właściwości RepeatBehavior

Dostępnych jest kilka opcji pozwalających na kontrolowanie zachowania animacji po zakończeniu jej odtwarzania. Wyboru dokonuje się za pomocą właściwości RepeatBehavior. Właściwość ta może przyjmować wartości trzech różnych typów:

* Czas podany w sekundach. Po upływie podanego czasu od zakończenia animacji, animacja zostanie odtworzona od początku.
* Tekst Forever. Animacja będzie powtarzana ciągle.
* Liczba zakończona literą x, określająca liczbę powtórzeń animacji. Na przykład chcąc, by animacja została odtworzona trzy razy, należy podać 3x.

Kompletny kod XAML animacji przedstawiającej prostokąt, trzykrotnie przesuwany wzdłuż osi X od współrzędnej 100 do 500 i z powrotem do 100, przedstawiono na **ilustracji 6**.

** Ilustracja 6. Odwracanie i powtarzanie animacji**

Skopiuj kod

<Rectangle x:Name="rect" Fill="Red"

 Canvas.Top="100" Canvas.Left="100"

 Width="100" Height="100">

 <Rectangle.Triggers>

 <EventTrigger RoutedEvent="Rectangle.Loaded">

 <BeginStoryboard>

 <Storyboard>

 <DoubleAnimation RepeatBehavior="3x"

 Storyboard.TargetName="rect"

 Storyboard.TargetProperty="(Canvas.Left)"

 To="500" Duration="0:0:5"

 AutoReverse="True" />

 </Storyboard>

 </BeginStoryboard>

 </EventTrigger>

 </Rectangle.Triggers>

</Rectangle>

Przyjrzyjmy się teraz bliżej poszczególnym typom animacji. Najpierw omówimy atrybuty potrzebne do uruchomienia animacji poszczególnych typów, a później dowiemy się, jak tworzyć animacje oparte na klatkach kluczowych.

Animowanie wartości z wykorzystaniem obiektu DoubleAnimation

Obiekt DoubleAnimation pozwala na określenie sposobu zmian wartości typu double w czasie. Animowana wartość w poszczególnych momentach wyznaczana jest na podstawie interpolacji liniowej wartości początkowej i końcowej.

Animując wartość typu double, należy określić wartość początkową za pomocą atrybutu From oraz wartość końcową za pomocą albo atrybutu To (określającego bezwzględną wartość końcową), albo atrybutu By (określającego względną zmianę wartości). Na przykład chcąc zmienić właściwość Canvas.Left jakiegoś obiektu od wartości 100 (blisko lewej krawędzi ekranu) do wartości 500, należy w atrybucie From podać wartość 100 i użyć albo atrybutu To z wartością 500, albo atrybutu By z wartością 400. Jeżeli ustawione zostaną oba atrybuty, pod uwagę zostanie wzięty atrybut To, a atrybut By zostanie zignorowany. Ponadto, jeżeli prostokąt znajduje się już w pozycji określonej przez wartość From, nie ma potrzeby określania wartości atrybutu From.

Sytuacja ta występuje w ostatnim przykładzie XAML. Właściwość Canvas.Left prostokąta ma wartość 100, a w obiekcie DoubleAnimation atrybut To ma wartość 500. Animacja spowoduje więc zmianę tej wartości od 100 do 500, czego efektem będzie przesunięcie prostokąta w prawo.

Animowanie koloru z wykorzystaniem obiektu ColorAnimation

Obiekt ColorAnimation działa w sposób podobny do DoubleAnimation. Stosowany jest do określania sposobu zmiany koloru elementu w czasie. Wartości koloru w poszczególnych chwilach wyznaczane są na podstawie interpolacji liniowej wartości początkowej i końcowej koloru.

Animując kolor, należy za pomocą atrybutu From określić wartość początkową. Jeżeli wartość początkowa nie zostanie określona, użyty zostanie kolor bieżący. Wartość docelową koloru należy określić za pomocą atrybutu To. Można także zastosować atrybut By, określający kolor docelowy jako sumę wartości koloru From (lub koloru początkowego) i koloru By.

Animując właściwość określającą kolor, nie definiuje się tej właściwości bezpośrednio jako właściwości docelowej, ponieważ właściwość ta zwykle definiuje obiekt Brush, a nie kolor. W związku z tym, chcąc animować na przykład kolor wypełnienia prostokąta, jako właściwość docelową należy podać nie właściwość Fill tego prostokąta, a właściwość Color obiektu SolidBrush, wykorzystywanego do wypełnienia tego prostokąta.

Sposób animacji koloru prostokąta przedstawiono na **ilustracji 7**. W ciągu 5 sekund kolor czarny zmieniany jest na kolor biały. Jak można zauważyć, właściwością docelową animacji jest właściwość Color obiektu SolidColorBrush, wykorzystywanego do wypełnienia prostokąta. Warto też zwrócić uwagę na składnię XAML wykorzystywaną do wskazywania właściwości złożonych.

** Ilustracja 7. Animowanie zmiany koloru**

Skopiuj kod

<Rectangle x:Name="rect" Canvas.Top="100"

 Canvas.Left="100" Width="100"

 Height="100" Fill="Black">

 <Rectangle.Triggers>

 <EventTrigger RoutedEvent="Rectangle.Loaded">

 <BeginStoryboard>

 <Storyboard>

 <ColorAnimation Storyboard.TargetName="rect"

 Storyboard.TargetProperty=

 "(Shape.Fill).(SolidColorBrush.Color)"

 To="#00000000" Duration="0:0:5" />

 </Storyboard>

 </BeginStoryboard>

 </EventTrigger>

 </Rectangle.Triggers>

</Rectangle>

Animowanie punktu z wykorzystaniem obiektu PointAnimation

Animacja typu PointAnimation służy do animowania wartości definiowanych jako punkty. Współrzędne punktu w poszczególnych chwilach wyznaczane są na podstawie interpolacji liniowej pozycji początkowej i końcowej.

Analogicznie jak w przypadku animacji koloru i wartości typu Double, pozycję początkową określa się za pomocą atrybutu From, a pozycję końcową albo za pomocą atrybutu By (względem pozycji początkowej), albo za pomocą atrybutu To (pozycja absolutna). Przykład animacji punktu końcowego krzywej Beziera przedstawiono na **ilustracji 8**. Krzywa Beziera jest zdefiniowana za pomocą punktu początkowego o współrzędnych (100, 100), punktu końcowego o współrzędnych (300, 100) oraz punktu kontrolnego o współrzędnych (200, 0). Animacja rozpoczyna się natychmiast po załadowaniu obiektu Path i powoduje przesunięcie punktu końcowego krzywej (Point2) ze współrzędnych (300, 100) na współrzędne (300, 600) w ciągu 5 sekund.

** Ilustracja 8. Animacja końca krzywej**

Skopiuj kod

<Path Stroke="Black" >

 <Path.Data>

 <PathGeometry>

 <PathFigure StartPoint="100,100">

 <QuadraticBezierSegment x:Name="seg"

 Point1="200,0" Point2="300,100" />

 </PathFigure>

 </PathGeometry>

 </Path.Data>

 <Path.Triggers>

 <EventTrigger RoutedEvent="Path.Loaded">

 <BeginStoryboard>

 <Storyboard>

 <PointAnimation Storyboard.TargetName="seg"

 Storyboard.TargetProperty="Point2"

 From="300,100" To="300,600" Duration="0:0:5" />

 </Storyboard>

 </BeginStoryboard>

 </EventTrigger>

 </Path.Triggers>

</Path>

Stosowanie klatek kluczowych

Poznane przed chwilą trzy typy animacji — ColorAnimation, DoubleAnimation i PointAnimation — zmieniają wskazaną właściwość w oparciu o interpolację liniową. Na przykład animacja wartości typu Double od wartości 100 do wartości 500 w ciągu 5 sekund powoduje, że wartość ta rośnie w tempie 80 na sekundę.

Dla każdego z trzech wymienionych typów animacji możliwe jest zdefiniowanie tempa zmian z wykorzystaniem zbioru punktów pośrednich, nazwanych klatkami kluczowymi. Aby zmodyfikować liniowy postęp animacji od wartości początkowej do końcowej, wystarczy zdefiniować jedną lub więcej klatek kluczowych i określić styl animacji, jaki chcemy zastosować pomiędzy poszczególnymi punktami.

Klatki kluczowe są definiowane na podstawie czasu. Czas określany jest względem początku animacji i definiuje moment wystąpienia klatki kluczowej. Chcąc zdefiniować na przykład 9-sekundową animację z trzema równomiernie rozłożonymi klatkami kluczowymi, pierwszą klatkę należy umieścić w chwili 0:0:3, drugą w 0:0:6, a trzecią w 0:0:9. W tym miejscu warto zwrócić uwagę, że klatka kluczowa nie ma czasu trwania — ma tylko swój czas wystąpienia.

Jako kolejny przykład rozważmy animację wartości typu Double, którą chcemy podzielić w połowie zakresu od 100 do 500. Pierwsza połowa animacji powinna zostać zrealizowana bardzo szybko, a druga — bardzo wolno. Czas trwania całej animacji powinien wynosić 6 sekund. Ponieważ połowa zakresu od 100 do 500 to 350, definiujemy klatkę kluczową o początku w punkcie 350. Aby pierwsza połowa dystansu została pokonana w ciągu jednej sekundy, klatkę te umieszczamy w chwili 0:0:1, a punkt końcowy definiujemy za pomocą drugiej klatki kluczowej w chwili 0:0:6. Teraz animowany element przemknie przez połowę odległości i będzie się wlókł przez pozostałą część dystansu.

W poprzednim przykładzie pozycje pośrednie w obu segmentach animacji są wyznaczane w oparciu o interpolację liniową. Aby zapewnić większą elastyczność, przewidziano dwa inne typy klatek kluczowych — klatkę dyskretną, powodującą natychmiastowy przeskok pomiędzy dwiema wartościami, i klatkę spline powodującą, że pozycje pośrednie są interpolowane w oparciu o linię drugiego stopnia. W kolejnych przykładach przedstawiono definiowanie klatek kluczowanych dla animacji wartości typu Double, ale takie same zasady obowiązują także dla animacji wartości typu Point i Color.

Aby zdefiniować animację opartą na klatkach kluczowych, należy do nazwy typu animacji dołączyć przyrostek UsingKeyFrames. W związku z tym, aby zdefiniować animację typu Double opartą na klatkach kluczowych, należy użyć obiektu DoubleAnimationUsingKeyFrames i w obiekcie tym wskazać docelowy obiekt i docelową właściwość animacji (w taki sam sposób, jak w obiekcie DoubleAnimation). Wewnątrz deklaracji obiektu DoubleAnimationUsingKeyFrames należy umieścić definicje klatek kluczowych. Jak już wcześniej pisałem, ta sama metoda dotyczy animacji PointAnimationUsingKeyFrames i ColorAnimationUsingKeyFrames.

Stosowanie liniowych klatek kluczowych

Domyślną metodą generowania animacji pomiędzy dwoma wartościami właściwości jest interpolacja liniowa, powodująca równomierne rozłożenie zmiany w czasie. Jeżeli czas animacji podzielimy na liniowe odcinki z wykorzystaniem typu LinearKeyFrame, animacja nadal będzie oparta na interpolacji liniowej, ale interpolacja będzie dotyczyła odcinków pomiędzy klatkami kluczowymi, co pozwala na uzyskanie efektu przyspieszenia lub opóźnienia.

Przeanalizujmy kod animacji przedstawiony na **ilustracji 9**. Zastosowano animację DoubleAnimationUsingKeyFrames i zdefiniowano dwie klatki kluczowe. Pierwsza powoduje zmianę właściwości Canvas.Left od wartości 0 do 300 w ciągu sekundy, a druga powoduje zmianę tej właściwości od wartości 300 do 600 w ciągu 8 sekund. Efektem jest szybkie przesunięcie prostokąta przez połowę dystansu, a następnie powolne przemieszczenie przez pozostałą odległość. Podobne zasady obowiązują dla LinearPointKeyFrame i LinearColorKeyFrame.

** Ilustracja 9. Animacja z wykorzystaniem klatek kluczowych**

Interpolacja liniowa

Skopiuj kod

<Rectangle Fill="#FFFF0000" Stroke="#FF000000"

 Width="40" Height="40"

 Canvas.Top="40" x:Name="rect">

 <Rectangle.Triggers>

 <EventTrigger RoutedEvent="Rectangle.Loaded">

 <BeginStoryboard>

 <Storyboard>

 <DoubleAnimationUsingKeyFrames

 Storyboard.TargetName="rect"

 Storyboard.TargetProperty="(Canvas.Left)" >

 <LinearDoubleKeyFrame KeyTime="0:0:1" Value="300" />

 <LinearDoubleKeyFrame KeyTime="0:0:9" Value="600" />

 </DoubleAnimationUsingKeyFrames>

 </Storyboard>

 </BeginStoryboard>

 </EventTrigger>

 </Rectangle.Triggers>

</Rectangle>

Dyskretne klatki kluczowe

Skopiuj kod

<Rectangle Fill="#FFFF0000" Stroke="#FF000000"

 Width="40" Height="40"

 Canvas.Top="40" x:Name="rect">

 <Rectangle.Triggers>

 <EventTrigger RoutedEvent="Rectangle.Loaded">

 <BeginStoryboard>

 <Storyboard>

 <DoubleAnimationUsingKeyFrames

 Storyboard.TargetName="rect"

 Storyboard.TargetProperty="(Canvas.Left)" >

 <DiscreteDoubleKeyFrame KeyTime="0:0:1" Value="300" />

 <DiscreteDoubleKeyFrame KeyTime="0:0:9" Value="600" />

 </DoubleAnimationUsingKeyFrames>

 </Storyboard>

 </BeginStoryboard>

 </EventTrigger>

 </Rectangle.Triggers>

</Rectangle>

Stosowanie dyskretnych klatek kluczowych

Chcąc zmienić właściwość z jednej wartości na inną bez stosowania interpolacji liniowej, można użyć dyskretnych klatek kluczowych. Spowoduje to, że właściwość obiektu zostanie zmieniona skokowo w chwili zdefiniowanej w klatce kluczowej.

W drugiej części **ilustracji 9** zamieszczono ten sam przykład z klatkami kluczowymi zamienionymi na klatki dyskretne. Sekundę po rozpoczęciu animacji prostokąt przeskoczy na środek ekranu, a po kolejnych 8 sekundach przeskoczy na prawą stronę. Analogiczne zasady obowiązują w przypadku animacji DiscretePointKeyFrame i DiscreteColorKeyFrame.

Stosowanie klatek kluczowych typu spline

Aby zmieniać właściwość w sposób płynny i umożliwiający przyspieszanie i zwalnianie, można użyć klatek kluczowych typu spline. W tym celu należy zdefiniować krzywą Beziera drugiego stopnia. Chwilowa szybkość zmian właściwości od wartości początkowej do końcowej wyznaczana jest przez rzut równoległy tej krzywej.

Jeżeli trudno jest to zrozumieć, wyobraźmy sobie następującą sytuację. Słońce świeci dokładnie w zenicie, a my wybijamy piłkę bejsbolową. Patrzymy na cień piłki. W czasie, gdy piłka wznosi się, cień wydaje się przyspieszać. Gdy piłka osiąga maksymalną wysokość, cień zwalnia, a gdy piłka opada — cień znowu przyspiesza.

Wróćmy do naszej animacji. Szybkość odtwarzania animacji odpowiada szybkości cienia piłki, a tor lotu piłki definiowany jest przez krzywą. Krzywą tą można zdefiniować za pomocą atrybutu KeySpline. Atrybut ten pozwala na zdefiniowanie punktów kontrolnych krzywej Beziera drugiego stopnia. Krzywa jest znormalizowana, co oznacza, że zaczyna się w punkcie 0, a kończy w punkcie 1. W przypadku łuku parabolicznego, odpowiadającego torowi lotu piłki, atrybut KeySpline będzie zawierał dwie rozdzielone przecinkami wartości znormalizowane.

Aby zdefiniować krzywą odpowiadającą torowi lotu piłki, należy użyć atrybutu KeySpline z wartością 0.3,0 0.6,1. Pierwszy punkt krzywej będzie miał współrzędne (0,3, 0), a drugi (0,6, 1). W efekcie animacja będzie szybko przyspieszała do mniej więcej jednej trzeciej toru lotu piłki, następnie będzie kontynuowała z mniej więcej stałą prędkością do około dwóch trzecich trajektorii lotu piłki, a przez pozostały okres będzie zwalniała, symulując spadanie piłki na ziemię.

Przykład zastosowania KeySpline do definicji krzywej na potrzeby tej symulacji z wykorzystaniem DoubleAnimationUsingKeyFrames przedstawiono na **ilustracji 10**. W przykładzie tym obiekt Ellipse jest animowany tak, by przemieszczał się po ekranie w sposób naśladujący ruch cienia piłki, tak jakbyśmy znajdowali się nad boiskiem i patrzyli w dół na piłkę lecącą w powietrzu.

** Ilustracja 10. Definicja krzywej z użyciem KeySpline**

Skopiuj kod

<Ellipse Fill="a#FF444444" Stroke="#FF444444"

 Width="40" Height="40"

 Canvas.Top="40" x:Name="ball">

 <Ellipse.Triggers>

 <EventTrigger RoutedEvent="Ellipse.Loaded">

 <BeginStoryboard>

 <Storyboard>

 <DoubleAnimationUsingKeyFrames

 Storyboard.TargetName="ball"

 Storyboard.TargetProperty="(Canvas.Left)" >

 <SplineDoubleKeyFrame KeyTime="0:0:5"

 KeySpline="0.3,0 0.6,1" Value="600" />

 </DoubleAnimationUsingKeyFrames>

 </Storyboard>

 </BeginStoryboard>

 </EventTrigger>

 </Ellipse.Triggers>

</Ellipse>

Animacja a Expression Blend

Animację można zdefiniować w sposób graficzny za pomocą Expression Blend™. narzędzie to pozwala na wygenerowanie kodu XAML animacji z automatycznym zastosowaniem odpowiednich typów animacji.

W Expression Blend w menu Window wybieramy polecenie Animation Workspace. Uzyskamy dostęp do narzędzi do graficznego projektowania linii czasu. Po wyedytowaniu w edytorze graficznym właściwości, które mają być modyfikowane przez animację, wygenerowany zostanie kod XAML animacji.

W dolnej części ekranu dostępny jest widok obiektów i linii czasu. Pozwala on na dodanie linii czasu i graficzne definiowanie klatek kluczowych. Aby dodać nową linię czasu, należy kliknąć przycisk + w widoku Objects and Timeline.

Gdy klikniemy ten przycisk, pojawi się okno dialogowe Create Storyboard, w którym należy podać nazwę tworzonego obiektu Storyboard. W naszym przypadku zmieniłem domyślną nazwę Storyboard1 na Timeline1 i usunąłem zaznaczenie pola wyboru Create as Resource.

Expression Blend pozwala na definiowanie animacji wewnątrz obiektu Canvas albo jako zasób. W tym pierwszym przypadku animacja może być uruchamiana w odpowiedzi na wyzwalacze w obiekcie Canvas. Poniższy fragment kodu to kod XAML utworzony w Expression Blend poprzez użycie okna dialogowego Create Storyboard i wskazanie, że animacja nie powinna zostać utworzona jako zasób.

Skopiuj kod

<Canvas.Triggers>

 <EventTrigger RoutedEvent="Canvas.Loaded">

 <BeginStoryboard>

 <Storyboard x:Name="Timeline1"/>

 </BeginStoryboard>

 </EventTrigger>

</Canvas.Triggers>

Gdybyśmy zaznaczyli pole Create as Resource, obiekt Storyboard zostałby utworzony wewnątrz obiektu <Canvas.Resources> i uruchomienie animacji wymagałoby użycia kodu JavaScript.

Utworzona linia czasu zostanie wyświetlona w widoku Objects and Timeline. Można ją zobaczyć w dolnej części **ilustracji 11**. Pionowa linia w chwili 0, widoczna na linii czasu, oznacza bieżący czas (w Expression Blend linia ta ma kolor żółty). Aby dodać klatkę kluczową, wystarczy przeciągnąć tę linię na pozycję, w której chcemy utworzyć klatkę kluczową, i kliknąć przycisk Record Keyframe. Przycisk ten znajduje się tuż ponad linią czasu i na lewo od tekstu 0:00:000.



Ilustracja 11. **Widok Animation Workspace w Expression Blend** (aby powiększyć ilustrację, kliknij ją)

Przeciągnijmy tę linię do znacznika czwartej sekundy i dodajmy klatkę kluczową. Klatka będzie widoczna jako niewielki owal na linii czasu. Po przewinięciu linii czasu do czwartej sekundy i dodaniu klatki kluczowej możemy zmodyfikować kolor, położenie, przezroczystość lub kształt prostokąta, a Expression Blend automatycznie wyznaczy odpowiednie transformacje. Przesuwając znacznik na linii czasu możemy uzyskać podgląd animacji i sprawdzić, jak ekran będzie wyglądał w określonej chwili.

Podsumowanie

W artykule tym omówiliśmy definiowanie transformacji i animacji w Silverlight XAML. Poznaliśmy różne typy transformacji pozwalających na obracanie, skalowanie i pochylanie obiektu, a także stosowanie dowolnych transformacji definiowanych za pomocą macierzy przekształcenia afinicznego. Następnie poznaliśmy podstawowe typy animacji i dowiedzieliśmy się, jak zdefiniować animację opartą na wyzwalaczu XAML. Sprawdziliśmy, w jaki sposób animacje zmieniają właściwości obiektów w czasie i zapoznaliśmy się z elementami XAML pozwalającymi na animowanie wartości liczbowych, punktowych i koloru. Następnie dowiedzieliśmy się, w jaki sposób używać klatek kluczowych w celu uzyskania bardziej precyzyjnej kontroli nad animacjami. Na koniec przekonaliśmy się, jak łatwo można generować animacje za pomocą Expression Blend.

**Laurence Moroney** pracuje w Microsoft na stanowisku Senior Technology Evangelist i specjalizuje się w technologii Silverlight. Jest autorem wielu książek informatycznych na tematy związane z Silverlight, AJAX, interoperacyjnością i bezpieczeństwem. Blog Laurence'a znajduje się pod adresem [blogs.msdn.com/webnext](http://blogs.msdn.com/webnext).

© 2008 Microsoft Corporation i CMP Media, LLC. Wszelkie prawa zastrzeżone. Powielanie w całości lub w części bez uprzedniej zgody jest zabronione.