Instituto Tecnológico de Nuevo Laredo

"Con la ciencia por la humanidad."

Ingeniería en Sistemas Computacionales

Table of Contents

Unidad 1

XAML

Contenedores

Figuras

Brochas solida

Unidad 2

Tipos de brocha

Unidad 3

Transformaciones

Unidad 4

StoryBoard y sus propiedades

General

Esta unidad se aprendió sobre las formas de utilizar xaml, manejando las figuras establecidas en el lenguaje, como la forma de adaptarlas a nuestras necesidades modificando sus propiedades.

También se vio sobre las maneras de animar una figura, a través de StoryBoarding y transformaciones de las figuras o proyecciones en un plano

¿Qué es XAML?

Es un lenguaje de marcado basado en XML desarrollado por Microsoft. XAML es el lenguaje que subyace a la presentación visual de una aplicación desarrollada en Microsoft Expression Blend, al igual que HTML es el lenguaje que subyace la presentación visual de una página web. La creación de una aplicación en Expression Blend supone tener que escribir código XAML, ya sea de forma manual o visual, mediante la vista Diseño de Expression Blend.

Contenedores

Canvas: Este contenedor permite mostrar elementos hijos en coordenadas específicas.
Grid: Este contenedor permite crecer filas y columnas para mostrar elementos hijos.
StackPanel: Este contenedor permite mostrar elementos secundarios en una línea, ya sea horizontal o verticalmente.
ScrollViewer: Este contenedor permite desplazarse para mostrar un elemento hijo que tiene una superficie superior.

Figuras

XAML proporciona varios objetos Shape listos para usar. Todos los objetos de formas heredan de la clase Shape. Los objetos de formas disponibles incluyen Ellipse, Line, Path, Polygon, Polyline y Rectangle.

Ellipse: Dibuja una elipse

Line: Dibuja una línea entre dos puntos

Path: Dibuja una serie de líneas y curvas conectadas.

Polygon: Dibuja un polígono, que es una serie conectada de líneas que constituyen una forma cerrada.

Polyline: Dibuja una serie de líneas rectas conectadas.

Rectangle: Dibuja un rectángulo.

Los objetos Shape comparten las siguientes propiedades comunes.

- Stroke : describe cómo se pinta el contorno de la forma.
- StrokeThickness : describe el grosor del contorno de la forma.
- Fill : describe cómo se pinta el interior de la forma.
- Propiedades de datos para especificar coordenadas y vértices, medidos en píxeles independientes del dispositivo.

Brochas

Se utilizan los objetos Brush para dibujar un área con un color sólido, un degradado o una imagen. Para pintar un objeto en la pantalla, como un Shape o un Control, se utiliza un Brush. Estableces la propiedad Fill del Shape o las propiedades Background y Foreground de un Control a un Brush deseado.

Los diferentes tipos de brochas en Silverlight para Windows Phone son SolidColorBrush, LinearGradientBrush, RadialGradientBrush e ImageBrush. Para obtener más información consulta Brochas en MSDN.

Brochas de color sólido

Un SolidColorBrush pinta un área con un solo Color, como rojo o azul. Hay tres formas en el XAML para definir un SolidColorBrush y el color que indica. Puedes usar los nombres predefinidos para los colores, valores hexadecimales o la sintaxis de elementos para propiedades.

Nombres predefinidos de color

Puedes utilizar un nombre predefinido de color, como Yellow o SlateBlue, ya sea para establecer la propiedad Fill de un Shape o las propiedades Background y Foreground de un Control. El intérprete de XAML convierte el nombre del color a una estructura Color con los canales de color correctos.

Valores hexadecimales de color

Puedes utilizar el valor hexadecimal del color para establecer la propiedad Fill de un Shape o las propiedades Background y Foreground de un Control. La estructura del valor hexadecimal es el canal alfa (opacidad), canal rojo, canal verde y el canal azul. Por ejemplo, el valor hexadecimal #FFFF0000 define el rojo completamente opaco (alfa = FF, FF = rojo, verde = 00, y azul = 00).

En el siguiente ejemplo se establece la propiedad Fill de un rectángulo al valor hexadecimal #FFFF0000.

XAML

```
<StackPanel>
                <Rectangle Width="100" Height="100" Fill="#FFFF0000 />
</ StackPanel>
```

Sintaxis de elementos para propiedades

Puedes utilizar la sintaxis de elementos para propiedades para definir un SolidColorBrush. Esta sintaxis es más elaborada que los métodos anteriores, pero te permite especificar opciones adicionales como el Opacity de una brocha.

Brochas

Introducción a los pinceles

Utiliza Brush objetos para pintar una área con un color sólido, un degradado o una imagen. Si está familiarizado con la forma pinceles trabajan en aplicaciones de Windows Phone que se ejecuta en el navegador, entonces usted estará encantado de saber que los cepillos funcionan exactamente de la misma manera en el Windows Phone. La única excepción es que VideoBrush no es compatible con el Windows Phone.

Para pintar un objeto en la pantalla, como una Forma o un control , utilice un cepillo de . Se establece el relleno propiedad de la Forma o el fondo y de primer plano propiedades de un control a la deseada Brush .

Los diferentes tipos de pinceles en Windows Phone son SolidColorBrush , LinearGradientBrush , RadialGradientBrush y ImageBrush .

Pinceles de colores sólidos

A SolidColorBrush pinta un área con un solo color , como el rojo o el azul. Hay tres maneras en XAML para definir un SolidColorBrush y el color se especifica. Puede usar nombres predefinidos de colores, los valores de color hexadecimales, o la sintaxis de elementos de propiedad.

Predefinidos nombres de colores

Puede utilizar un nombre de color predefinido, como Amarillo o SlateBlue, a bien fijar el relleno propiedad de un Shape o la de fondo y de primer plano propiedades de un control . El analizador XAML convierte el nombre de color a un color de la estructura con los canales de color correctos.

El ejemplo siguiente establece el relleno propiedad de un rectángulo con el color predefinido Roja .

```
XAML
<StackPanel>
      <Rectángulo width = "100" height = "100" Fill = "Red" />
</ StackPanel>
```

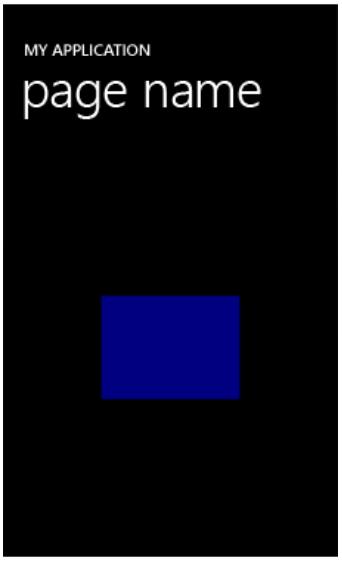
Valores de color hexadecimales

Usted puede utilizar el valor hexadecimal del color a bien fijar el relleno propiedad de un Shape o la de fondo y de primer plano propiedades de un control . La estructura del valor hexadecimal es el canal alfa (opacidad), canal rojo, el canal verde y el canal azul. Por ejemplo, el valor hexadecimal # FFFF0000define rojo totalmente opaco (alpha = FF, FF = rojo, verde = 00, y azul = 00).
El ejemplo siguiente establece el relleno propiedad de un rectángulo para el valor hexadecimal # FFFF0000.

```
XAML
<StackPanel>
   <rectángulo width = "100" height = "100" Fill = "# FFFF0000" />
</ StackPanel>
```

Sintaxis Propiedad del elemento

Usted puede utilizar la sintaxis de elementos de propiedad para definir un SolidColorBrush . Esta sintaxis es más prolija que los métodos anteriores, pero que le permite especificar opciones adicionales, tales como el cepillo de la opacidad . Para obtener más información sobre la sintaxis XAML, incluida la sintaxis de elementos de propiedad, consulte Inicio rápido: Creación de una interfaz de usuario con XAML para Windows Phone .
El ejemplo siguiente se crea un rectángulo y explícitamente crea el SolidColorBrush para el relleno propiedad. El color del SolidColorBrush se establece en azul y la opacidad se establece en 0,5.

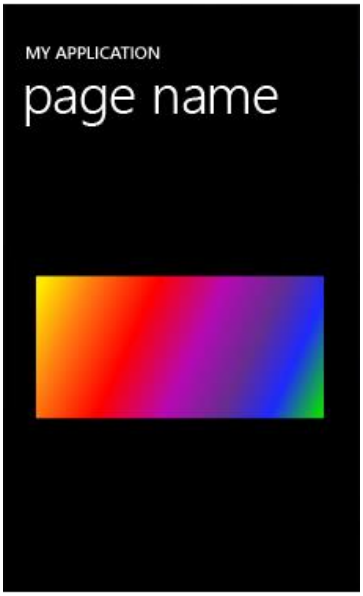



```
XAML
<Grid >
  <Rectangle Width = " 200 " Height = " 150 " >
     <Rectangle.Fill >
       <SolidColorBrush Color = " Blue " Opacity = " 0.5 " />
     </ Rectangle.Fill >
  </ Rectangle >
</ Grid >
```

Degradado lineal

A LinearGradientBrush pinta un área con un degradado que se define a lo largo de una línea, llamado el eje de degradado. Especifique los colores del degradado y su ubicación a lo largo del eje de degradado utilizando GradientStop objetos. También puede modificar el eje de degradado, lo que le permite crear gradientes horizontales y verticales, y para invertir la dirección del gradiente.

El GradientStop es el componente básico de un pincel de degradado edificio. Un punto de degradado especifica un color a una compensación a lo largo del eje de degradado.
Del punto de degradado de color propiedad especifica el color del punto de degradado. Puede ajustar el color mediante el uso de un nombre de color predefinido o especificando los valores ARGB hexadecimales.
Del punto de degradado Offset propiedad especifica la posición del color de la parada de degradado en el eje de degradado. El Offset es un doble que va de 0 a 1. El valor de desplazamiento más cerca de un punto de degradado es 0, más cerca del color es al inicio del gradiente. El valor de compensación de la pendiente es más cerca a 1, más cerca del color es hasta el final del gradiente.
El siguiente ejemplo crea un gradiente lineal con cuatro colores y la utiliza para pintar un rectángulo .



XAML

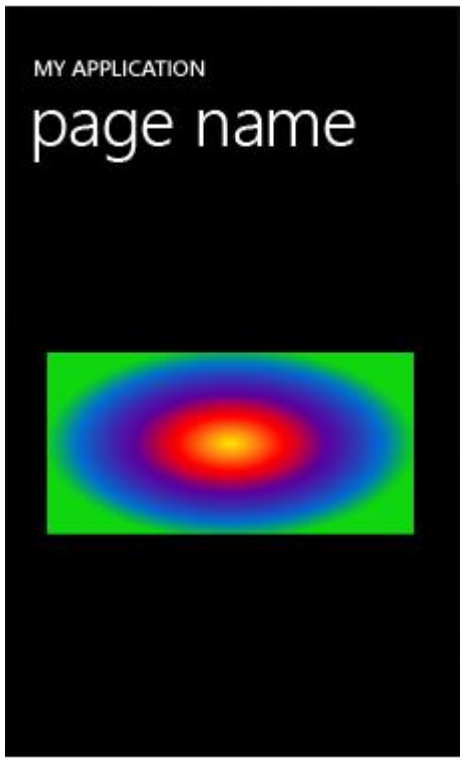
```
<StackPanel>
  <- Este rectángulo se pinta con un degradado lineal diagonal.  ->
      <Rectangle href = "400" = "200">
             <Rectangle.Fill>
                <LinearGradientBrush StartPoint = "0,0" EndPoint = "1,1">
                     <GradientStop color = "Yellow" Offset = "0,0" />
                     <GradientStop color = "Red" Offset = "0.25" />
                     <GradientStop color = "Blue" Offset = "0.75" />
                     <GradientStop color = "LimeGreen" Offset = "1,0" />
                 </ LinearGradientBrush>
              </ Rectangle.Fill>
      </ Rectangle>
</ StackPanel>
```

Degradado radial

Como un LinearGradientBrush , un LinearGradientBrush pinta un área con colores que se entremezclan a lo largo de un eje. Eje de un pincel de degradado radial está definida por un círculo; sus colores irradian hacia fuera desde su origen. Utilice la GradientOrigin , Centro , RadiusX Y RadiusY propiedades, para definir el degradado radial.

XAML

```
<StackPanel>
    <- Este rectángulo se pinta con un degradado radial. -->
   <Rectangle Width = " 400 " Height = " 200 " >
       <Rectangle.Fill >
                    <RadialGradientBrush GradientOrigin = " 0.5,0.5 " Center = "
                    0.5,0.5 " RadiusX = " 0.5 " RadiusY = " 0.5 " >
                     <GradientStop Color = " Yellow " Offset = " 0 " />
                      <GradientStop Color = " Red " Offset = " 0.25 " />
                      <GradientStop Color = " Blue " Offset = " 0.75 " />
                       <GradientStop Color = " LimeGreen " Offset = " 1 " />
                      </ RadialGradientBrush >
         </ Rectangle.Fill >
    </ Rectangle >
</ StackPanel >
```



Transformaciones

XAML contiene un conjunto de clases que permiten aplicar transformaciones a elementos en 2 dimensiones que nos permiten modificar el tamaño y la posición de los elementos, todas estas clases derivan de System.Windows.Media.Transform.

Todos los elementos del tipo System.Windows.Controls.FrameworkElement tienen dos propiedades de tipo System.Windows.Media.Transform que puede usarse para aplicar transformaciones:

LayoutTransfrom

La transformación se procesa antes de emitir el resultado del XAML a pantalla.

RenderTransfrom

La transformación se procesa después de emitir el resultado del XAML a pantalla.

Además los elementos de tipo System.Windows.UIElement, poseen una propiedad RenderTransformOrigin que se utiliza para indicar el punto en que comenzara la transformación.

La propiedad RenderTransformOrigin, es de tipo System.Windows.Point, por tanto, un origen puede tomar los siguientes valores :

- (0,0) Esquina superior izquierda de un elemento
- (0,1) Esquina inferior izquierda de un elemento
- (1,0) Esquina superior derecha de un elemento
- (1,1) Esquina inferior derecha de un elemento

También, podemos usar números decimales, pero los valores siempre deben estar comprendidos entre 0 y 1, de este modo, un origen de (0.5, 0.5) representaría el medio del elemento.

La transformación RotateTransform rota un elemento acorde a la propiedad Angle que especifica los grados aplicados en la rotación, este tipo de transformación ha sido la que hemos utilizado en el ejemplo anterior para rotar el botón central 45 grados.

La transformación ScaleTransform, estira o encoje un elemento de modo vertical, horizontal o de ambos modos mediante el uso de las propiedades ScaleX y ScaleY.

Un valor de 0.5 para la propiedad ScaleX, hace que un elemento se encoja a la mitad de su anchura, y un valor de 2 hace que el elemento se estire hasta alcanzar el doble de su anchura original.

Cuando hace uso de ScaleY, lo que hacemos es afectar a la altura del elemento en función del valor recibido, un valor de 0.5 hace que el elemento se encoja hasta alcanzar la mitad de la altura original, un valor de 2 hace que el elemento se estire hasta alcanzar el doble de su altura original.

La transfromación SkewTransfrom inclina un elemento de acuerdo a los valores de las propiedades AngleX y AngleY que especifican los grados de inclinación aplicados a cada eje.

Para terminar, existe la posibilidad de combinar varias transformaciones a la vez sobre un elemento, de tal modo que podemos aplicar rotación y escalado simultáneamente, y no solo eso, si no que podemos hacer uso tanto de transformaciones RenderTransform como de transformaciones LayoutTransform.

Para aplicar varias transformaciones a la vez, debemos hacer uso de la clase TransformGroup, y envolver dentro de la misma las transformaciones que deseemos aplicar sobre el elemento.

Storyboard y sus propiedades

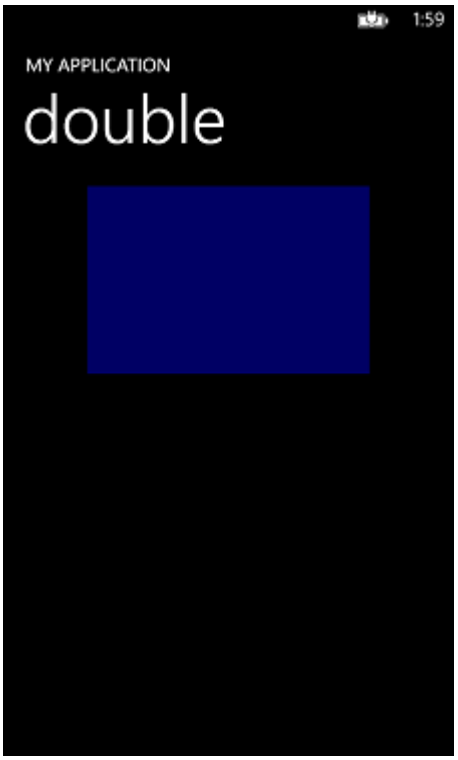
Animación de una propiedad doble

Animaciones de Windows Phone se crean al cambiar los valores de propiedades de los objetos. Por ejemplo, puede animar el ancho de un rectángulo , el ángulo de una RotateTransform , o el valor de color de un botón .

En el siguiente ejemplo, la opacidad propiedad está animado.

XAML

```
<StackPanel>
   <StackPanel.Resources>
      <- Anima la opacidad del rectángulo.  ->
      <Storyboard x: Name = "myStoryboard">
         <DoubleAnimation
         Storyboard.TargetName = "MyAnimatedRectangle"
         Storyboard.TargetProperty = "opacidad"
         De = "1.0" a = "0.0" Duración = "0:0:1"
         AutoReverse = "true" />
      </ Storyboard>
   </ StackPanel.Resources>
   <Rectangle MouseLeftButtonUp = "Rectangle_Tapped"
      x: Name = "MyAnimatedRectangle"
      Width = "300" height = "200" Fill = "Blue" />
</ StackPanel>
```



Las secciones siguientes describen los pasos para animar la propiedad Opacity y examinan el XAML que se utiliza para cada paso.

1. La identificación de la propiedad para animar

En este ejemplo, desea animar la opacidad característica de un rectángulo . Usted no tiene que declarar la propiedad que desea animar en el objeto en sí. Sin embargo, normalmente el nombre del objeto que desea animar. Nombrar el objeto hace que sea más fácil para especificar qué objeto está siendo el blanco de la animación. El siguiente XAML muestra cómo denominar el rectánguloMyAnimatedRectangle.

XAML

```
<Rectangle x: Name = "MyAnimatedRectangle" ...
```

2. Creación de un guión gráfico y por lo que es un recurso

Un guión gráfico es el contenedor que se pone objetos de animación en. Usted tiene que hacer elStoryboard un recurso que está disponible para el objeto que desea animar. El siguiente XAML muestra cómo hacer el Storyboard un recurso del elemento raíz, que es un StackPanel .

XAML

```
<StackPanel x: Name = "rootElement">
   <StackPanel.Resources>
   <- Anima la opacidad del rectángulo.  ->
      <Storyboard x: Name = "myStoryboard">
         <- Objetos de animación van aquí.  ->
      </ Storyboard>
   </ StackPanel.Resources>
</ StackPanel>
```

3. Adición de un objeto de animación al guión gráfico

Debido a que el valor de la propiedad que está animando (Opacidad) utiliza un doble, este ejemplo se utiliza el DoubleAnimation objeto. Un objeto de animación especifica lo que está animado y cómo se comporta esa animación. El siguiente XAML muestra cómo la DoubleAnimation se agrega al guión gráfico .

XAML

```
<Storyboard x: Name = "myStoryboard">
    <DoubleAnimationStoryboard.TargetName="MyAnimatedRectangle"
Storyboard.TargetProperty="Opacity" From="1.0" To="0.0" Duration="0:0:1"
AutoReverse="True" RepeatBehavior="Forever" /> </ Storyboard>
```

Esta DoubleAnimation objeto especifica la siguiente animación:

- Storyboard.TargetProperty = "opacidad" especifica que la opacidad propiedad está animado.
- Storyboard.TargetName = especifica "MyAnimatedRectangle", que se oponen a esta propiedad está animando (el rectángulo).
- De = "1.0" a = "0,0" indica que la opacidad propiedad comienza en un valor de 1 y anima a 0 (comienza opaco y luego se desvanece).
- Duración = "0:0:1" especifica la duración de la animación (lo rápido que el rectángulo se desvanece). Debido a que la Duración propiedad se especifica en forma de "horas: minutos: segundos", la duración utilizada en este ejemplo es de un segundo.
- AutoReverse = "true" especifica que cuando termina la animación, que va a la inversa. En el caso de este ejemplo, que se desvanece y luego se invierte para una opacidad completa.
- RepeatBehavior = "Forever" especifica que cuando se inicia la animación, continúa indefinidamente. En este ejemplo, el rectángulo se desvanece en y fuera de forma continua.

4. Inicio de la animación

Una forma común para iniciar una animación es en respuesta a un evento. En este ejemplo, elMouseLeftButtonUp evento se utiliza para iniciar la animación cuando el usuario toca el rectángulo .

XAML

```
<Rectangle MouseLeftButtonUp = "Rectangle_Tapped"
   x: Name = "MyAnimatedRectangle"
   Width = "100" height = "100" Fill = "Blue" />
```

El guión gráfico se inicia mediante la Empiece método.

```
myStoryboard.Begin ();
```

Usted puede utilizar C # o Visual Basic en lugar de XAML para hacer una animación.

Animación de una propiedad de color

El ejemplo anterior muestra cómo animar una propiedad que utiliza un valor del doble . ¿Qué pasa si usted quiere animar un color ? Teléfono de Windows proporciona objetos de animación que se utilizan para animar otros tipos de valores. Los siguientes objetos básicos de animación animan propiedades del doble , color y punto , respectivamente:

- DoubleAnimation
- ColorAnimation
- PointAnimation

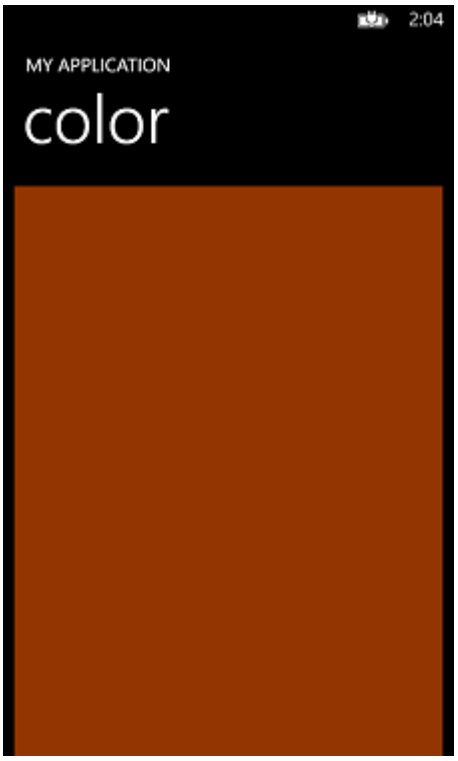
Nota:

También puede animar propiedades que utilizan objetos.

El siguiente ejemplo muestra cómo crear un ColorAnimation .
XAML

```
<StackPanel MouseLeftButtonUp = "Rectangle_Tapped">
   <StackPanel.Resources>
      <Storyboard x: Name = "myStoryboard">
      <- Animar el color de fondo de la tela del rojo al verde de más de 4 segundos.
->
         <ColorAnimation Storyboard.TargetName = "mySolidColorBrush"
            Storyboard.TargetProperty = "Color"
            De = "Red" To = Duración "Green" = "0:0:4" />
      </ Storyboard>
   </ StackPanel.Resources>
   <StackPanel.Background>
      <SolidColorBrush x: Name = "mySolidColorBrush" Color = "Red" />
   </ StackPanel.Background>
</ StackPanel>
```

C #



Iniciar, detener, pausar y reanudar

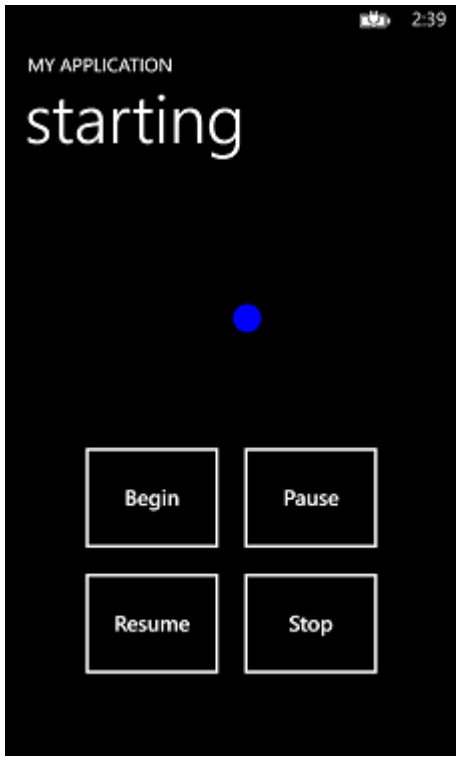
El ejemplo anterior muestra cómo iniciar una animación utilizando la Empiece método. Storyboardtambién tiene Detener , Pausa y Reanudar métodos que se pueden utilizar para controlar una animación. El ejemplo siguiente crea cuatro botones objetos que permiten al usuario controlar la animación de una elipse en la pantalla.

XAML

```
<Lienzo>
  <Canvas.Resources>
    <Storyboard x: Name = "myStoryboard">
    <-! Animar el punto central de la elipse.  ->
    <PointAnimation Storyboard.TargetProperty = "Centro"
      Storyboard.TargetName = "MyAnimatedEllipseGeometry"
      Duración = "0:0:5"
      De = "20200"
      Para = "400100"
      RepeatBehavior = "Forever" />
    </ Storyboard>
  </ Canvas.Resources>
  <Rellenar trazado = "Blue">
    <Path.Data>
    <- Describir una elipse.  ->
      <EllipseGeometry x: Name = "MyAnimatedEllipseGeometry"
      Centro = "20,20" RadiusX = "15" RadiusY = "15" />
    </ Path.Data>
  </ Path>
  <StackPanel Orientación = "Vertical" Canvas.Left = "60" Canvas.Top = "265">
   <StackPanel Orientación = "horizontal">
    <-! Button que inicia la animación.  ->
    <Button Click = "Animation_Begin"
      Width = "165" height = "130" Margen = "2" content = "Begin" />
    <-! Button que se detiene la animación.  ->
    <Button Click = "Animation_Pause"
      Width = "165" height = "130" Margen = "2" content = "Pausa" />
   </ StackPanel>
   <StackPanel Orientación = "horizontal">
    <-! Button que se reanuda la animación.  ->
    <Button Click = "Animation_Resume"
      Width = "165" height = "130" Margen = "2" content = "Reanudar" />
    <- Botón que detiene la animación.  Detener la animación devuelve el
    elipse a su ubicación original.  ->
```

```
      <Button Click = "Animation_Stop"
        Width = "165" height = "130" Margen = "2" content = "Stop" />
    </ StackPanel>
  </ StackPanel>
</ Canvas>
```

C #



La animación utilizando fotogramas clave

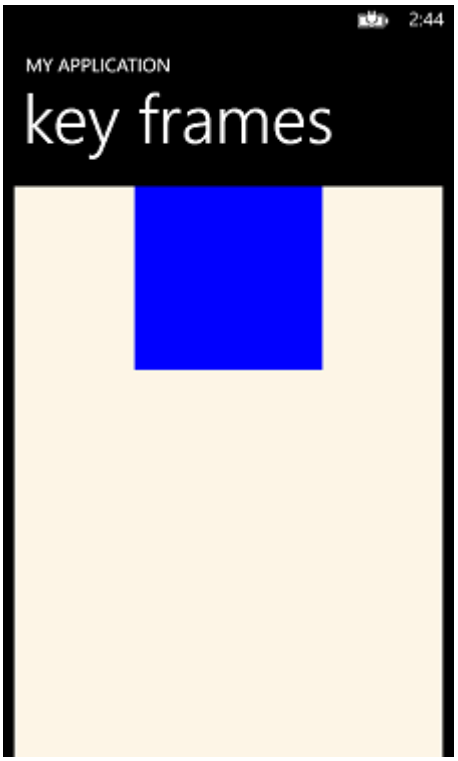
Hasta ahora, los ejemplos de este inicio rápido han demostrado la animación de entre dos valores.(Estos se llaman From / To / By animaciones.) Animaciones de fotogramas clave permiten utilizar más de dos valores de destino y controlar el método de interpolación de una animación. Al especificar múltiples valores para animar, puede hacer animaciones más complejas. Al especificar la interpolación de la animación (en concreto, mediante la propiedad KeySpline), se puede controlar la aceleración de una animación.
El siguiente ejemplo muestra cómo utilizar una animación de fotogramas clave para animar la altura de un rectángulo .

XAML

```
< StackPanel Background = " #FDF5E6 " > < StackPanel.Resources > < Storyboard
x:Name = " myStoryboard " > < DoubleAnimationUsingKeyFrames
Storyboard.TargetName = " myRectangle " Storyboard.TargetProperty = " Height " >
<!--  Este fotograma clave restablece la animación a su valor inicial (30) al comienzo
```

```
de la animación.  -> <LinearDoubleKeyFrame Value = "30" KeyTime = "0:0:0" /> <-! Animaciones Spline se utilizan para crear la aceleración.  Esta SplineDoubleKeyFrame crea una animación que empieza lento y luego acelera.  El rectángulo "cae".  -> <SplineDoubleKeyFrame KeySpline = "0,0 1,0" Value = "300" KeyTime = "0:0:0.8" /> <- Esta animación spline crea el "rebote" en el extremo donde se acorta rectángulo su longitud rápidamente al principio y luego se ralentiza y se detiene.  -> <SplineDoubleKeyFrame KeySpline = "0.10, 0.21 0.00, 1.0 " Value = " 250 " KeyTime = " 0:0:1.5 " /> </ DoubleAnimationUsingKeyFrames > </ Storyboard > </ StackPanel.Resources > < Rectangle x:Name = " myRectangle " MouseLeftButtonUp = " Rectangle_Tapped " Fill = " Blue " Width = " 200 " Height = " 30 " /> </ StackPanel >
```

C #



El XAML incluye los siguientes tres fotogramas clave. Cada fotograma clave especifica un valor para animar a en un momento determinado. Toda la animación tarda 1,5 segundos.

- El primer fotograma clave en este ejemplo es un LinearDoubleKeyFrame .LinearTypeKeyFrame objetos como LinearDoubleKeyFrame crear una transición suave y lineal entre los valores. Sin embargo, en este ejemplo, que sólo se utiliza para especificar que la animación es a valor 30 en el tiempo 0.
- El segundo fotograma clave en este ejemplo es un SplineDoubleKeyFrame , que especifica que laaltura del rectángulo es 300 en tiempo de 0,8 segundos

después de que comience la animación.SplineTypeKeyFrame objetos como SplineDoubleKeyFrame crean una transición variable entre los valores según el valor de la KeySpline propiedad. En este ejemplo, el rectángulo comienza a moverse lentamente y luego acelera hacia el final del segmento de tiempo.

- El tercer fotograma clave en este ejemplo es un SplineDoubleKeyFrame , que especifica que laaltura del rectángulo es de 250 en el momento de 1,5 segundos después de que comience la animación (0,7 segundos después de la última SplineDoubleKeyFrame terminó). En contraste con el anterior SplineDoubleKeyFrame , este fotograma clave hace que la animación de inicio de rápido y lento hacia el final.

Quizás la propiedad más complicada que utiliza el SplineDoubleKeyFrame es la KeySpline propiedad.Esta propiedad especifica los primero y segundo puntos de control de una curva de Bezier, que describe la aceleración de la animación.

Animar utilizando funciones de aceleración

Funciones de aceleración permiten aplicar fórmulas matemáticas personalizadas a sus animaciones. Por ejemplo, es posible que desee un objeto bote con realismo, ni se comporta como si fuera un muelle.Usted podría utilizar fotogramas clave o incluso From / To / By animaciones para aproximar estos efectos, pero se necesitaría una cantidad significativa de trabajo y la animación sería menos preciso que el uso de una fórmula matemática.

Además de la creación de su propia función de aceleración personalizada mediante la implementación del IEasingFunction interfaz, puede utilizar una de varias funciones de aceleración que proporciona el tiempo de ejecución para crear efectos comunes. El ejemplo siguiente muestra las funciones de aceleración que vienen con el tiempo de ejecución. Seleccione una función de aceleración de la lista desplegable, establecer sus propiedades, a continuación, ejecute la animación. El XAML para la animación se muestra en la parte inferior derecha del ejemplo.

A continuación se muestra una lista de las funciones de aceleración demostrado en el ejemplo anterior, junto con un breve resumen de lo que hace.

Función de aceleración	Resumen
BackEase	Se retrae el movimiento de una animación un poco antes de que comience a animar en el camino indicado.
BounceEase	Crea un efecto de rebote.
CircleEase	Crea una animación que acelera y / o desacelera mediante una función circular.
CubicEase	Crea una animación que acelera y / o desacelera mediante la fórmula f (t) = t3.
ElasticEase	Crea una animación que se asemeja a una oscilación del resorte de ida y vuelta hasta que se detiene.
ExponentialEase	Crea una animación que acelera y / o desacelera mediante una fórmula exponencial.
PowerEase	Crea una animación que acelera y / o desacelera mediante la fórmula f (t) = TP, donde p es igual a la propiedad de alimentación.
QuadraticEase	Crea una animación que acelera y / o desacelera mediante la fórmula f (t) = t2.
QuarticEase	Crea una animación que acelera y / o desacelera mediante la fórmula f (t) = t4.
QuinticEase	Crear una animación que acelera y / o desacelera mediante la fórmula f (t) = t5.
SineEase	Crea una animación que acelera y / o desacelera mediante una fórmula sinusoidal.

Puede aplicar estas funciones de aceleración para las animaciones de fotogramas clave utilizando
cualquiera EasingDoubleKeyFrame , EasingPointKeyFrame o EasingColorKeyFrame . El

siguiente ejemplo muestra cómo utilizar fotogramas clave con funciones de aceleración asociados con ellos para crear una animación de un rectángulo que se contrae hacia arriba, se ralentiza, luego se expande hacia abajo (como si la caída) y luego rebota a una parada.

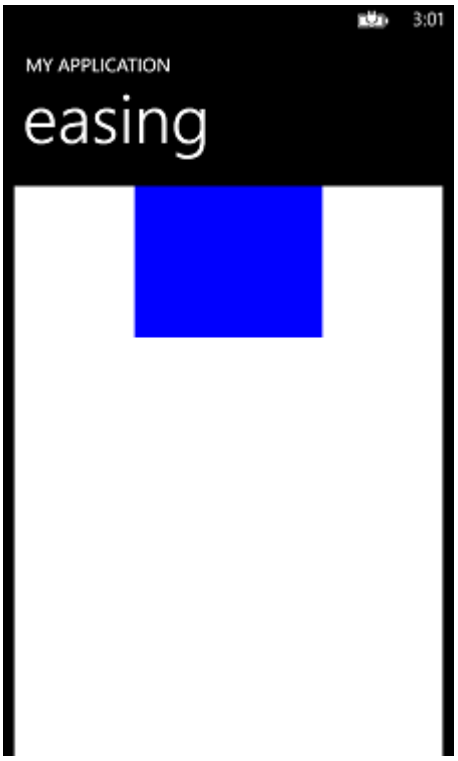
XAML

```
<StackPanel Background = "White">
   <StackPanel.Resources>
      <Storyboard x: Name = "myStoryboard">
         <DoubleAnimationUsingKeyFrames
         Storyboard.TargetProperty = "Altura"
         Storyboard.TargetName = "myRectangle">

         <- Este fotograma clave anima la elipse hasta la cresta
          donde se ralentiza y se detiene.  ->
         <EasingDoubleKeyFrame Value = "30" KeyTime = "00:00:02">
            <EasingDoubleKeyFrame.EasingFunction>
            <CubicEase EasingMode = "velocidadFinal" />
         </ EasingDoubleKeyFrame.EasingFunction>
         </ EasingDoubleKeyFrame>

         <- Este fotograma clave anima la elipse hacia abajo y hace que rebote.  ->
         <EasingDoubleKeyFrame Value = "200" KeyTime = "00:00:06">
            <EasingDoubleKeyFrame.EasingFunction>
            <BounceEase Bounces = "5" EasingMode = "velocidadFinal" />
         </ EasingDoubleKeyFrame.EasingFunction>
         </ EasingDoubleKeyFrame>
         </ DoubleAnimationUsingKeyFrames>
      </ Storyboard>
   </ StackPanel.Resources>
   < Rectangle x:Name = " myRectangle " MouseLeftButtonUp = " Rectangle_Tapped
" Fill = " Blue " Width = " 200 " Height = " 200 " />
 </ StackPanel>
```

C #



Además de utilizar las funciones de aceleración incluidas en el tiempo de ejecución, puede crear su propia costumbre funciones de aceleración heredando de EasingFunctionBase .

Conclusiones y recomendaciones

Hay distintas maneras de modificar los controles de nuestra aplicación, y aunque muchas veces existe la necesidad de utilizar métodos mucho muy precarios, XAML nos da la facilidad de hacerlo de una manera más sencilla y útil haciendo que nuestra programación se concentre en las cosas más importantes como la lógica de negocios.