Лабораторная работа №8.

Анимации и пользовательские элементы управления WPF

Цель: Изучить основы создания анимированных приложений WPF.

Задание

1. Реализовать анимированных переход между окнами в приложении.
2. Реализовать несколько видов анимаций по различным свойствам зависимостей.
3. Реализовать собственный элемент управления.

Краткие теоретические сведения.

Анимации WPF

Анимация — оптическая иллюзия, которая достигается частой сменой статических изображений с целью создания иллюзии движения. В программировании ранее для реализации анимации необходимо было выполнить следующие действия для анимации (создания динамики статических объектов):

1. Создать таймер
2. Проверять значение таймера в дискретные интервалы времени
3. Выполнять анимирующее действие в эти дискретные интервалы времени, возможно с вычислением нового значения того или иного свойства (положения относительно левого края экрана, прозрачности фона и т. п.)
4. Перерисовывать поверхность с новыми значениями.

В технологии WPF анимация является «first-class citizen», т. е. одной из основных концепций, внедренных в платформу. При помощи технологии WPF относительно просто создавать сложные анимации с использованием множества разных свойств, триггеров и с подключением множества одновременно работающих таймеров. Самое главное преимущество для программиста в данном случае — декларативное описание анимаций; программист не задумывается о необходимых программных служебных структурах для поддержки анимации, он лишь указывает, какую конкретно анимацию и каким образом ему необходимо достичь.

Анимация и свойства

Прежде всего, необходимо понимать, что анимация относится к свойства. Например, для увеличения объекта, можно использовать его height и width свойства. Для перемещения по экрану можно использовать его свойства Margin или Canvas.Left (в случае, если он нарисован на канве).

Для поддержки анимации, свойства должны иметь следующие характеристики:

- Свойство должно быть зависимым (dependency property);
- Оно должно быть внутри класса, реализующего DependencyObject и Ianimatable;
- Должен существовать хотя бы один совместимый тип анимации (имеется возможность создавать собственные типы анимации).

Простейший пример анимации, использующий DoubleAnimation (линейно изменяющийся набор значений по времени):

```
<Window x:Class="WpfApplication1.MainWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      Title="MainWindow" Height="350" Width="525">
   <Grid>
      <StackPanel Margin="10">
         <Rectangle
            Name="MyRectangle"
            Width="100"
            Height="100"
            Fill="Blue">
            <Rectangle.Triggers>
```

```
            <!-- Animates the rectangle's opacity. -->
            <EventTrigger RoutedEvent="Rectangle.Loaded">
              <BeginStoryboard>
                <Storyboard>
                  <DoubleAnimation
                    Storyboard.TargetName="MyRectangle"
                    Storyboard.TargetProperty="Opacity"
                    From="1.0" To="0.0" Duration="0:0:5"
                    AutoReverse="True" RepeatBehavior="Forever" />
                </Storyboard>
              </BeginStoryboard>
            </EventTrigger>
          </Rectangle.Triggers>
        </Rectangle>
      </StackPanel>
    </Grid>
  </Window>
```

В данном случае прямоугольник будет исчезать и появляться (при помощи изменения его прозрачности) бесконечно каждую секунду.

Типы анимаций

Анимация представляет собой просто набор значений, применяемых к свойству, сгенерированных на промежутке времени. Из-за того, что свойства имеют различные типы, существуют и различные типы анимаций. Например, для анимирования свойств, имеющих тип Double, необходимо использовать DoubleAnimation; для Point — PointAnimation. Помимо этого, суещствуют различные типы в зависимости от того, какой числовой ряд генерируется. Рассмотрим некоторые из них.

- <Тип>Animation — линейно-изменяющаяся анимация, характеризуется тремя свойствами: от, до, шаг. (From/To/By animation)

- <Тип>AnimationUsingKeyFrames — анимация, похожая на From/ToBy анимацию, но имеющую несколько конечных точек. Т.е. вместо анимирования от конкретного значения до конкретного значения, анимирование происходит в несколько точек, например: от 0 до 50 за 3 секунды; от 50 до 20 за 2 секунды; от 20 до 100 за 4 секунды. Таким образом можно добиваться «прыгающих» или «скользящих» анимаций.

- <Тип>AnimationUsingPath — анимация, указываемая при помощи пути (Path).

- <Тип>AnimationBase — базовый класс для реализации классов

-<Тип>Animation <Тип>AnimationUsingKeyFrames. Используется только при реализации собственных анимаций.

StoryBoard

Прежде всего, каждая анимация наследуется от объекта Timeline. Timline представляет собой некоторый промежуток времени. Данные промежуток характеризуется длительностью, повторимостью и скоростью его прохождения (он не обязательно привязан к системным часам). Всякая анимация применяется к свойству конкретного элемента управления. Для этого у класса StoryBoard (доска истории; аггрегат анимаций) содержатся свойства TargetName и TargetProperty — целевой элемент управления и целевое свойство.

Сами по себе StoryBoard поддерживают также возможность контроля над анимацией — остановку, паузу, перемещение по временной оси (timeline) и другое. StoryBoard можно запустить при помощи EventTrigger'a (как показано в примере выше) или при помощи StoryBoard.Begin метода в коде. При этом у каждой StoryBoard есть возможность подписаться на события начала и окончания и ряд других.

При использовании анимаций с изменяемыми параметрами, задаваемыми при помощи привязок (Binding), анимации необходимо перезапускать — в силу того, что необходимо регенерировать числовой ряд.

Элементы управления WPF

Традиционно для множества технологий, используемых в программировании, существует два способа создания собственных элементов управления:

- Композитные элементы управления - UserControl;
- Собственные элементы управления — Custom Controls.

Композитные элементы управления представляют собой набор совмещенных существующих в платформе элементов управления, связанных логически одной целью. Создание таких элементов не отличается ничем от создания собственных приложений, поэтому этот способ рассматриваться не будет.

Выбор базового класса

При создании собственных элементов управления имеются следующие возможности:

- Наследование от UserControl — это способ создания композитного элемента управления; такой способ создания собственных является традиционно наиболее простым, однако наиболее ограниченным: отсутствует поддержка стилизации элемента управления при помощи DataTemplate или ControlTemplate, код нагроможден в одном месте и может конфликтовать с другими элементами управления и другое.
- Наследование от Control — данный способ избавлен от недостатков предыдущего и позволяет (и поощряет) использовать шаблоны элементов управления, стили для декорирования и поддержки различных тем оформления элементов управления.
- Наследование от FrameworkElement — самый низкоуровневый способ реализации собственного элемента управления; в данном случае возможен вариант собственной реализации отображения (рендеринга) элемента управления на экране, таким образом имея наибольшую гибкость (и наибольшую сложность) в реализации.

Основой для содержания логики настройки элементов управления являются зависимые свойства (dependency properties). Они предоставляют следующую функциональность:

- Поддержку механизма привязок;
- Поддержку стилей, триггеров;
- Поддержку механизмов валидации, обновления значений, установки пороговых значений и другое;
- Поддержку анимаций;
- Использование динамических ресурсов в качестве источников данных для этих свойств;
- И другое.

Для создания собственного зависимого свойства, выполните следующее:

```
/// <summary>
/// Identifies the Value dependency property.
/// </summary>
public static readonly DependencyProperty ValueProperty =
    DependencyProperty.Register(
        "Value", typeof(decimal), typeof(NumericUpDown),
        new FrameworkPropertyMetadata(MinValue, new
PropertyChangedCallback(OnValueChanged), new
CoerceValueCallback(CoerceValue)));
/// <summary>
/// Gets or sets the value assigned to the control.
/// </summary>
public decimal Value
{
    get { return (decimal)GetValue(ValueProperty); }
    set { SetValue(ValueProperty, value); }
}
private static object CoerceValue(DependencyObject element, object value)
```

```
{
    decimal newValue = (decimal)value;
    NumericUpDown control = (NumericUpDown)element;
    newValue = Math.Max(MinValue, Math.Min(MaxValue, newValue));
    return newValue;
}
private static void OnValueChanged(DependencyObject obj,
DependencyPropertyChangedEventArgs args)
{
    NumericUpDown control = (NumericUpDown)obj;
    RoutedPropertyChangedEventArgs<decimal> e = new
RoutedPropertyChangedEventArgs<decimal>(
        (decimal)args.OldValue, (decimal)args.NewValue, ValueChangedEvent);
    control.OnValueChanged(e);
}
```

Для каждого зависимого свойства определяется статический регистратор, само свойство, функция вызываемая при изменении значения и функция корректировки значения (для установки порогового).

Помимо зависимых свойств, ключевым также является использование переадресуемых событий (Routed events). Прежде всего, это необходимо для поддержания общей концепции обработки событий в WPF, а также для поддержки механизмов EventTrigger/EventSetter. Для создания переадресуемого события, выполните следующее:

```
/// <summary>
/// Identifies the ValueChanged routed event.
/// </summary>
public static readonly RoutedEvent ValueChangedEvent =
EventManager.RegisterRoutedEvent(
    "ValueChanged", RoutingStrategy.Bubble,
    typeof(RoutedPropertyChangedEventHandler<decimal>), typeof(NumericUpDown));
/// <summary>
/// Occurs when the Value property changes.
/// </summary>
public event RoutedPropertyChangedEventHandler<decimal> ValueChanged
{
    add { AddHandler(ValueChangedEvent, value); }
    remove { RemoveHandler(ValueChangedEvent, value); }
}
/// <summary>
/// Raises the ValueChanged event.
/// </summary>
/// <param name="args">Arguments associated with the ValueChanged event.</param>
protected virtual void OnValueChanged(RoutedPropertyChangedEventArgs<decimal> args)
{
    RaiseEvent(args);
}
```

Дополнительная информация по созданию собственных элементов управления может быть найдена по следующим ссылкам:

http://msdn.microsoft.com/en-us/library/ee330302.aspx
http://msdn.microsoft.com/en-us/library/ms752339.aspx

Контрольные вопросы.

1. Как поддерживается анимация в технологии WPF?
2. Какие существуют варианты создания собственных элементов управления в технологии WPF ?