# Модель событий

## Маршрутизация событий

Ранее рассматривались различные элементы управления. Но чтобы с ними взаимодействовать, нам надо использовать модель событий. WPF в отличие от других технологий, например, от Windows Forms, предлагает новую концепцию событий - **маршрутизированные события** (**routed events**).

Для элементов управления в WPF определено большое количество событий, которые условно можно разделить на несколько групп:

* События клавиатуры
* События мыши
* События стилуса
* События сенсорного экрана/мультитач
* События жизненного цикла

### Подключение обработчиков событий

Подключить обработчики событий можно декларативно в файле xaml-кода, а можно стандартным способом в файле отделенного кода.

Декларативное подключение:

|  |  |
| --- | --- |
| 1 | <Button x:Name="Button1" Content="Click" Click="Button\_Click" /> |

И подключим еще один обработчик в коде, чтобы при нажатии на кнопку срабатывали сразу два обработчика:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | public partial class MainWindow : Window  {      public MainWindow()      {          InitializeComponent();          Button1.Click += Button1\_Click;      }      // обработчик, подключаемый в XAML      private void Button\_Click(object sender, RoutedEventArgs e)      {          MessageBox.Show("Hi from Button\_Click");      }      // обработчик, подключаемый в конструкторе      private void Button1\_Click(object sender, RoutedEventArgs e)      {          MessageBox.Show("Hi from Button1\_Click");      }  } |

### Определение маршрутизированных событий

Определение маршрутизированных событий отличается от стандартного определения событий в языке C#. Для определения маршрутизированных событий в классе создвалось статическое поле по типу **RoutedEvent**:

|  |  |
| --- | --- |
| 1 | public static RoutedEvent СобытиеEvent |

Это поле, как правило, имеет суффикс *Event*. Затем это событие регистрируется в статическом конструкторе.

И также класс, в котором создается событие, как правило определяет объект-обертку над обычным событием. В этой обертке с помощью метода **AddHandler** происходит добавление обработчика для данного события, а с помощью метода **RemoveHandler** - удаление обработчика.

К примеру, возьмем встроенные класс ButtonBase - базовый класс для всех кнопок, который определяет ряд событий, в том числе событие **Click**:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | public abstract class ButtonBase : ContentControl, ...  {      // определение событие      public static readonly RoutedEvent ClickEvent;        static ButtonBase()      {          // регистрация маршрутизированного события          ButtonBase.ClickEvent = EventManager.RegisterRoutedEvent("Click",              RoutingStrategy.Bubble, typeof(RoutedEventHandler), typeof(ButtonBase));          //................................      }      // обертка над событием      public event RoutedEventHandler Click      {          add          {              // добавление обработчика              base.AddHandler(ButtonBase.ClickEvent, value);          }          remove          {              // удаление обработчика              base.RemoveHandler(ButtonBase.ClickEvent, value);          }      }    // остальное содержимое класса  } |

Маршрутизированные события регистрируются с помощью метода EventManager.RegisterRoutedEvent(). В этот метод передаются последовательно имя события, тип события (поднимающееся, прямое, опускающееся), тип делегата, предназначенного для обработки события, и класс, который владеет этим событием.

### Маршрутизация событий

Модель событий WPF отличается от событий WinForms не только декларативным подключением. События, возникнув на одном элементе, могут обрабатываться на другом. События могут подниматься и опускаться по дереву элементов.

Так, маршрутизируемые события делятся на три вида:

* **Прямые** (direct events) - они возникают и отрабытывают на одном элементе и никуда дальше не передаются. Действуют как обычные события.
* **Поднимающиеся** (bubbling events) - возникают на одном элементе, а потом передаются дальше к родителю - элементу-контейнеру и далее, пока не достигнет наивысшего родителя в дереве элементов.
* **Опускающиеся, туннельные** (tunneling events) - начинает отрабатывать в корневом элементе окна приложения и идет далее по вложенным элементам, пока не достигнет элемента, вызвавшего это событие.

Все маршрутизируемые события используют класс **RoutedEventArgs** (или его наследников), который представляет доступ к следующим свойствам:

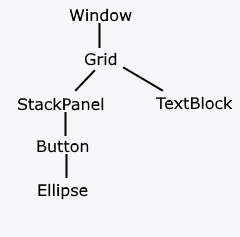
* **Source**: элемент логического дерева, являющийся источником события.
* **OriginalSource**: элемент визуального дерева, являющийся источником события. Обычно то же самое, что и Source
* **RoutedEvent**: представляет имя события
* **Handled**: если это свойство установлено в True, событие не будет подниматься и опускаться, а ограничится непосредственным источником.

### Поднимающиеся события

Допустим, у нас имеется такая разметка xaml:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <Window x:Class="EventsApp.MainWindow"          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"          xmlns:d="http://schemas.microsoft.com/expression/blend/2008"          xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"          xmlns:local="clr-namespace:EventsApp"          mc:Ignorable="d"          Title="MainWindow" Height="250" Width="400">      <Grid>          <Grid.ColumnDefinitions>              <ColumnDefinition Width="Auto" />              <ColumnDefinition />          </Grid.ColumnDefinitions>          <StackPanel Grid.Column="0" VerticalAlignment="Center" MouseDown="Control\_MouseDown">              <Button x:Name="button1" Width="80" Height="50" MouseDown="Control\_MouseDown" Margin="10" >                  <Ellipse Width="30" Height="30" Fill="Red" MouseDown="Control\_MouseDown" />              </Button>          </StackPanel>          <TextBlock x:Name="textBlock1" Grid.Column="1" Padding="10" />      </Grid>  </Window> |

В данном случае мы получаем следующее дерево элементов:

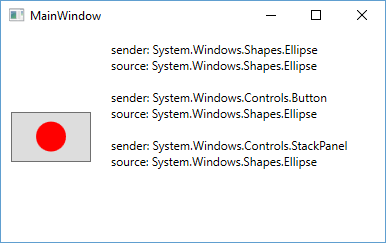


Три элемента имеют привязку к одному обработчику события, которое возникает при нажатии правой кнопки мыши или тачпада. Определим этот обработчик в файле кода C#:

|  |  |
| --- | --- |
| 1  2  3  4  5 | private void Control\_MouseDown(object sender, MouseButtonEventArgs e)  {      textBlock1.Text = textBlock1.Text + "sender: " + sender.ToString() + "\n";      textBlock1.Text = textBlock1.Text + "source: " + e.Source.ToString() + "\n\n";  } |

Обработчик в данном случае выводит информацию о событии в текстовый блок.

И так как это событие **MouseDown** является поднимающимся, то при нажатии правой кнопкой мыши на элемент самого нижнего уровня - Ellipse, событие MouseDown будет подниматься к контейнерам и отработает три раза последовательно для всех элементов Ellipse, Button, StackPanel:



### **Туннельные события**

Туннельные события действуют прямо противоположным способом. Как правило, все они начинаются со слова Preview. Возьмем выше приведенный пример и заменим событие MouseDown на PreviewMouseDown

|  |  |
| --- | --- |
| 1  2  3  4  5 | <StackPanel Grid.Column="0" VerticalAlignment="Center" PreviewMouseDown="Control\_MouseDown">      <Button x:Name="button1" Width="80" Height="50" PreviewMouseDown="Control\_MouseDown" Margin="10" >          <Ellipse Width="30" Height="30" Fill="Red" PreviewMouseDown="Control\_MouseDown" />      </Button>  </StackPanel> |

Нажмем на элемент Ellipse. Тогда событие сначала отработает на элементе StackPanel и затем последовательно на элементе Button и закончится на элементе Ellipse.

### C:\Users\KAB6-Teacher\Pictures\6.2.png

### **Прикрепляемые события (Attached events)**

Если у нас есть несколько элементов одного и того же типа и мы хотим привязать их к одному событию, то мы можем воспользоваться прикрепляемыми событиями.

Так, ранее у нас была группа элементов RadioButton и, чтобы вывести при выборе любого из них выбранное значение, нам приходилось у каждого определять обработчик события. Но это не оптимальная модель. И именно здесь мы и применим прикрепляемые события:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | <StackPanel x:Name="menuSelector" Grid.Column="0" RadioButton.Checked="RadioButton\_Click">      <RadioButton GroupName="menu" Content="Салат Оливье" />      <RadioButton GroupName="menu" Content="Котлета по-киевски" />      <RadioButton GroupName="menu" Content="Пицца с овощами" />      <RadioButton GroupName="menu" Content="Мясной рулет" />  </StackPanel> |

Обработчик для прикрепляемого события задается в формате *Имя\_класса.Название\_события="Обработчик"*. Здесь атрибут RadioButton.Checked="RadioButton\_Click" закрепляет все радиокнопки на StackPanel за одним обработчиком. Тогда в коде можно прописать:

|  |  |
| --- | --- |
| 1  2  3  4  5 | private void RadioButton\_Click(object sender, RoutedEventArgs e)  {      RadioButton selectedRadio = (RadioButton)e.Source;      textBlock1.Text = "Вы выбрали: " + selectedRadio.Content.ToString();  } |

И на текстовый блок выводится выбранный пункт.

Также обработчик для прикрепляемого события мы можем задать в коде c#:

|  |  |
| --- | --- |
| 1 | menuSelector.AddHandler(RadioButton.CheckedEvent, new RoutedEventHandler(RadioButton\_Click)); |