СОБЫТИЯ В WPF

Каждый элемент в **WPF** имеет ряд событий, на которые система может реагировать и обновлять содержимое.

Так как элементы формируются исходя из наследования от основных классов модели, каждый элемент наследует события от базовых классов.

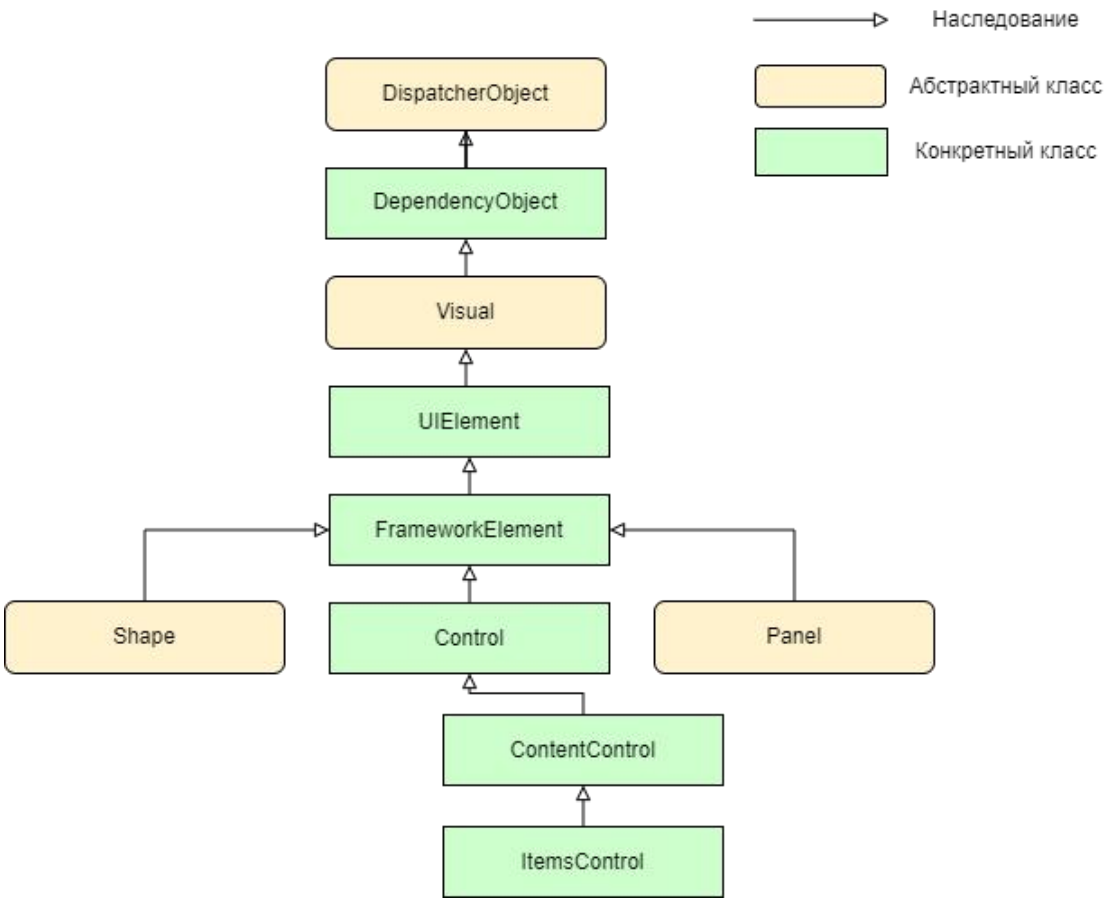


Рисунок 1

Первый класс, который вносит события элементу - **UIElement**. Он определяет базовые события ввода с клавиатуры, действия мыши и пера (для сенсорного ввода). Это значит, что подобными событиями обладает каждый элемент **WPF**.

> События являются некоторыми «триггерами», которые могут впоследствии хранить список методов, необходимые к выполнению после возникновения.

С помощью специальных свойств зависимости, можно привязать так называемые обработчики - пользовательские методы, выполняемые при возникновении определённого события.

Рисунок 2

Каждый обработчик реализует предписанный делегат. Обычно, это похоже на такую конструкцию.

Рисунок 3

Маршрутизация событий

Элементы в интерфейсе определяются древовидной структурой **XAML**. Как же быть, если два объекта, наложенных друг на друга, используют одно событие *(например, реагируют на движение мыши в своих пределах)*. Какое событие будет вызвано первым? В каком порядке события начнут выполняться?

Нужно всегда знать в каком порядке выполняются методы. **WPF** предлагает маршрутизируемые события. Что это означает?

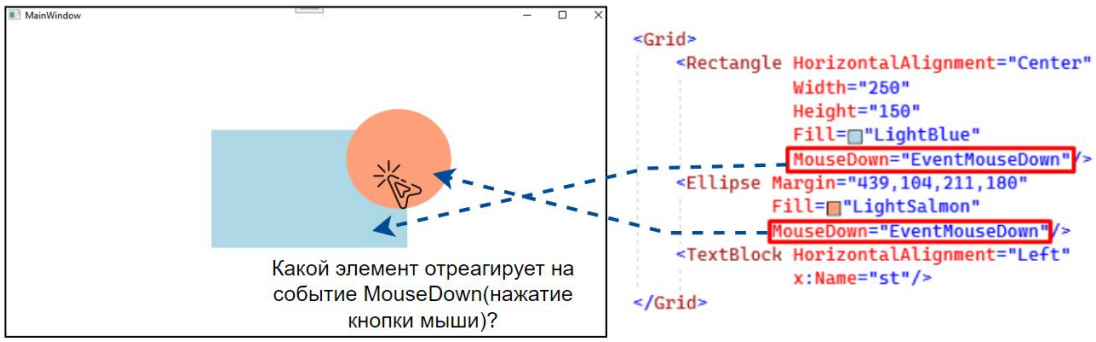


Рисунок 4

Для начала нужно понимать, какое дерево интерфейса было построено.

Если элементы находятся ***на одном уровне*** *(например, наложены друг на друга в одном контейнере)*, то события отработают ***на конкретном элементе***.

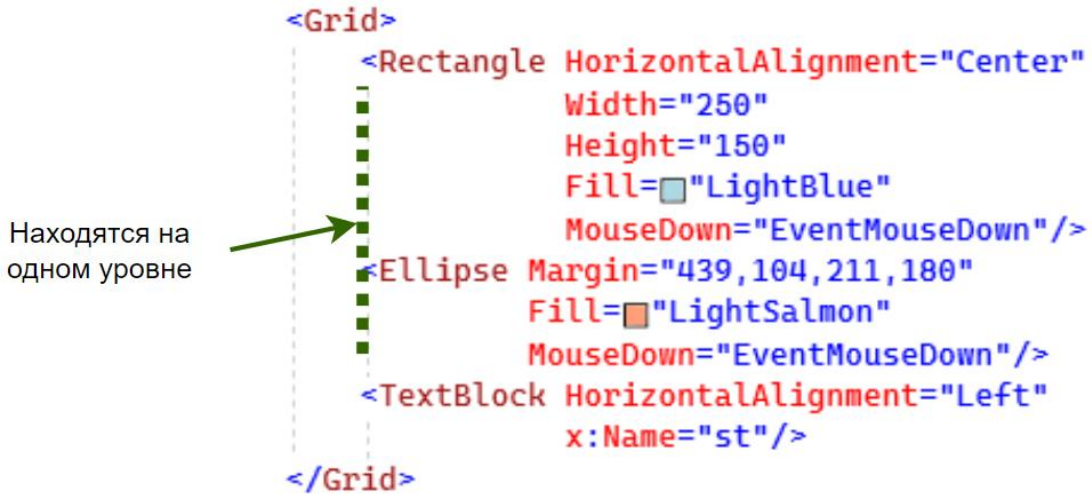
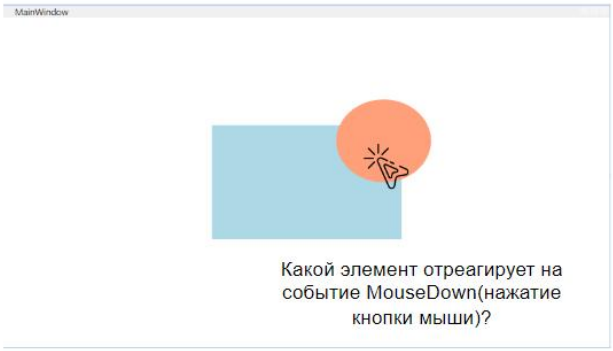


Рисунок 5

Если мы имеем дело с вложенными элементами, у которых обрабатывается одно и то же событие, появляется понятие маршрута.

> Нужно отметить, что все элементы в интерфейсе подвержены вложенной структуре. Такая структура называется «Иерархическим деревом». Для окна, корневым элементом является Window, а вложенным - Grid, и так далее, распространяясь будто ветки, формируется элемент.

Заменим в примере `Rectangle` на `Grid`, в который вложим `Ellipse`.




```
<Grid>
    <Grid HorizontalAlignment="Center"
              Width="250"
              Height="150"
              Background="LightBlue"
              MouseDown="EventMouseDown">
        <Ellipse
             Fill="LightSalmon"
             MouseDown="EventMouseDown" Margin="164,-34,-39,75"/>
    </Grid>
    <TextBlock HorizontalAlignment="Left"
              x:Name="st"/>
</Grid>
```

Маршрут события - это путь передачи уведомления о событии по ветке дерева интерфейса.

Маршрут события определяет само событие. Каждое событие может принимать один из трёх видов маршрутизации:

- ***Поднимающееся (пузырьковое)*** - событие возникает в исходном элементе. Далее уведомление о событии передаётся элементам до корневого*(родительского)* - ***вверх по дереву***.
- ***Опускающееся (туннельное)*** - возникает в исходном элементе и передаётся вложенным*(дочерним)* элементам - ***вниз по дереву***.
- ***Прямое (немаршрутизируемое)*** - на событие реагирует только тот элемент, на который направлено.

Событие `MouseDown`, рассматриваемое в примере, является поднимающимся. Если клик произошёл в точке вложенности, то первым отреагирует корневой, и в нашем примере корневым является `Grid`. После чего, `Grid` передаст событие `Ellipse`.

Абстрактно маршрутизацию можно представить на следующей схеме.

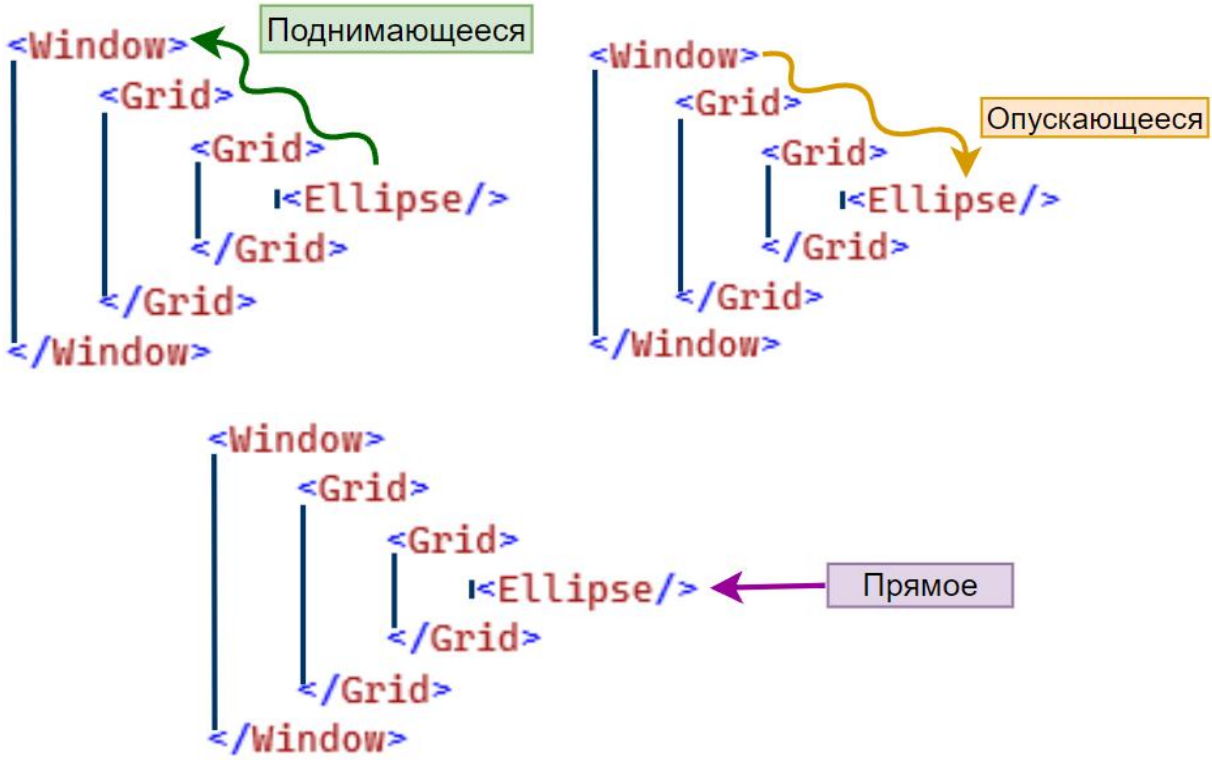


Рисунок 6

По началу легко запутаться, ведь маршрутизацию сложно рассматривать исходя из графического отображения. Однако мы имеем дело с разметкой **XAML**, где поднимающееся событие определяется такой иерархической структурой. Можно воспринимать элементы, как ветви общего корня.

Вот, например, картинка из учебника биологии за 6 класс *(игнорируем придаточные корни)*.

Рисунок 7

Аргументы маршрутизируемых событий

Каждое событие имеет специальный аргумент, который предоставляет дополнительную информацию о событии.

> Далее будут рассмотрены аргументы, которые наследуются от `RoutedEventArgs`. Соответственно, любой аргумент события будет иметь в себе свойства, предоставляемые этим типом.

Аргументы, предоставляемые обработчику событий запечатаны в тип `RoutedEventArgs`, из которого можно узнать много полезной информации. https://learn.microsoft.com/ru-ru/dotnet/api/system.windows.routedeventargs?view=windowsdesktop-7.0

У маршрутизируемого события совсем немного свойств.

Название свойства	Тип значения	Определение
`Source/OriginalSource`	object	Объект, вызвавший событие
`Handled`	bool	Определяет, передавать ли событие дальше по маршруту.
`RoutedEvent`	RoutedEvent	Возвращает данные о вызвавшем маршрутизируемом событии. В нём можно узнать имя события, владельца, а так же стратегию маршрута (один из трёх).

Мышь

Статический класс `Mouse` предоставляет доступ по управлению манипулятором типа мышь. Можно узнать, что это за девайс, а так же привязать или отвязать обработчики. Именно этот класс инициирует вызов всех связанных с мышью событий.

Базовые события мыши

Класс `UIElement` наследует события `Mouse` и предоставляет следующие виды событий мыши.

Название события	Тип маршрутизации	Триггер события
`PreviewMouseLeftButtonDown`	Опускающееся	Нажатие левой кнопки мыши.
`PreviewMouseRightButtonDown`	Опускающееся	Нажатие правой кнопки мыши.
`MouseLeftButtonDown`	Поднимающееся	Нажатие левой кнопки мыши.
`MouseRightButtonDown`	Поднимающееся	Нажатие правой кнопки мыши.
`PreviewMouseLeftButtonUp`	Опускающееся	Отпуск левой кнопки мыши.
`PreviewMouseRightButtonUp`	Опускающееся	Отпуск правой кнопки мыши.
`MouseRightButtonUp`	Поднимающееся	Отпуск правойкнопки мыши.
`MouseLeftButtonUp`	Поднимающееся	Отпуск левой кнопки мыши.
`MouseEnter`	Прямое	Мышь заходит в границы элемента
`PreviewMouseDown`	Опускающееся	Нажатие клавиши мыши.
`MouseDown`	Поднимающееся	Нажатие клавиши мыши.
`PreviewMouseUp`	Опускающееся	Отпуск клавиши мыши.
`MouseUp`	Поднимающееся	Отпуск клавиши мыши.
`PreviewMouseMove`	Опускающееся	Движение мыши в пределах элемента.
`MouseMove`	Поднимающееся	Движение мыши в пределах элемента.
`MouseLeave`	Прямое	Мышь выходит из границ элемента.
`GotMouseCapture`	Прямое	Получение фокуса на элемент от мыши.
`LostMouseCapture`	Прямое	Потеря фокуса элемента от мыши.
`PreviewMouseWhell`	Опускающееся	Движение колёсика мыши.
`MouseWhell`		Движение колёсика мыши.

Можно заметить, что у методов с приставкой Preview - опускающееся

Захват мыши

Иногда случаются такие задачи, в которых только определённому объекту необходимо реагировать на события мыши. В `WPF` есть возможность захватить мышь. Для этого вызывают метод `Mouse.Capture(<элемент>)`, где элемент представляет ссылку на тот элемент, которому необходимо «приватизировать» мышь. Для отмены захвата в качестве ссылки передают `null`.

Аргументы для обработчиков событий мыши

События, которые были вызваны мышью, передают обработчикам аргумент типа `MouseEventArgs`, который наследуется от `RoutedEventArgs`. https://learn.microsoft.com/ru-ru/dotnet/api/system.windows.forms.mouseeventargs?view=windowsdesktop-7.0

События мыши обычно инициируются объектом типа Mouse. Можно опредеить следующие наиболее важные свойства, которые передаёт аргумент `MouseEventArgs`.

Название свойства	Тип значения	Определение
`ButtonState`	Перечисление ***MouseButtonState***: Pressed и Released	Состояние нажания кнопки (опущена или поднята).
`ChangedButton`	Перечисление ***MouseButton***: Left Middle Right XButton1 XButton2	Кнопка, вызвавшая событие.
`LeftButton`	MouseButtonState	Возвращает состояние левой кнопки.
`MiddleButton`	MouseButtonState	Возвращает состояние средней кнопки.
`RightButton`	MouseButtonState	Возвращает состояние правой кнопки.
`XButton1`	MouseButtonState	Возвращает состояние первой дополнительной кнопки.
`XButton2`	MouseButtonState	Возвращает состояние второй дополнительной кнопки.

Так же стоит выделить метод `GetPosition(<элемент>)`, который возвращает объект `Point` с координатами, относительно элемента.

Клавиатура

Статический класс `Keyboard` отвечает за манипулятор типа клавиатура. Можно узнать, что это за девайс, а так же привязать или отвязать обработчики. Именно этот класс инициирует вызов всех связанных с клавиатурой событий.

● Базовые события клавиатуры

Класс `UIElement` наследует события `Keyboard` и предоставляет следующие виды событий клавиатуры.

Название события	Тип маршрутизации	Триггер события
`PreviewKeyDown`	Опускающееся	Нажатие клавиши
`KeyDown`	Поднимающееся	Нажатие клавиши
`PreviewKeyUp`	Опускающееся	Отпускание клавиши
`KeyUp`	Поднимающееся	Отпускание клавиши
`PreviewTextInput`	Опускающееся	Возникает в текстовом элементе при вводе текста
`TextInput`	Поднимающееся	Возникает в текстовом элементе при вводе текста
`GotKeyboardFocus`	Прямое	Получение фокуса клавиатуры
`LostKeyboardFocus`	Прямое	Потеря фокуса клавиатуры

● Аргументы обработчиков клавиатуры

В обработчики событий клавиатуры передаётся объект типа `KeyEventArgs`, который наследуется от `RoutedEventArgs`. https://learn.microsoft.com/ru-ru/dotnet/api/system.windows.input.keyeventargs?view=windowsdesktop-7.0

Можно выделить следующие наиболее интересные свойства этого класса.

Название свойства	Тип значения	Определение
`IsDown`	bool	Нажата ли кнопка
`IsRepeat`	bool	Удерживается ли кнопка
`IsToggled`	bool	Включена ли кнопка (для специальных клавиш типа CapsLock).
`IsUp`	bool	Отпущена ли кнопка
`Key`	Перечисление Key	Возвращает клавишу,

		которая вызвала событие.
`SystemKey`	Перечисление Key	Возвращает системную клавишу (та, что обрабатывается системой), которая вызвала событие.
`KeyStates`	Перечисление KeyStates: None Down Toggled	Состояние клавиши.

Drag'N'Drop события

Drag'N'Drop - это действие, подобное копированию и вставке данных, при котором мышь может перемещать некоторую информацию из одной области*(элемента)* в другую.

У некоторых элементов *(например, `TextBox`)* уже есть встроенная возможность переносить с помощью мыши выделенный текст.

Со стороны системы, перетаскивание упрощённо можно представить так: мышь инициирует событие `MouseDown` на одном объекте, а событие отпуска `MouseUp` на другом. WPF «из коробки» предоставляет следующие события элемента для работы с перетаскиванием, находящиеся в объекте `DragDrop`.

Название события	Тип маршрутизации	Триггер события
`DragEnter`	Поднимающееся	Вхождение мыши в элемент при перетаскивании.
`PreviewDragEnter`	Опускающееся	Вхождение мыши в элемент при перетаскивании.
`DragOver`	Поднимающееся	Движение в границах элемента при перетаскивании.
`PreviewDragOver`	Опускающееся	Движение в границах элемента при перетаскивании.
`DragLeave`	Поднимающееся	Покидание границ элемента при

		перетаскивании.
`PreviewDragLeave`	Опускающееся	Покидание границ элемента при перетаскивании.
`Drop`	Поднимающееся	Завершение перетаскивания.
`PreviewDrop`	Опускающееся	Завершение перетаскивания.

Все эти события направлены на целевой элемент перемещения. Для того, чтобы инициировать перемещение, необходимо использовать

Для того, чтобы разрешить элементу получать данные с помощью перетаскивания, необходимо установить `true` свойству `AllowDrop`.

Аргументы событий перетаскивания

Событиям перетаскивания в обработчик передаётся аргумент DragEventArgs.https://learn.microsoft.com/ru-ru/dotnet/api/system.windows.forms.drageventargs?view=windowsdesktop-7.0

Название свойства	Тип значения	Определение
`Effects`	Перечисление ***DragDropEffect***: All: данные копируются из источника в целевой элемент с удалением из источника Copy: данные просто копируются из источника в целевой элемент Link: данные из источника связываются с данными из целевого элемента Move: данные перемещаются из источника в целевой элемент None: отсутствие эффекта Scroll: данные прокручиваются при копировании в целевой	Задаёт действие при перетаскивании

	элемент	
AllowedEffect	Перечисление DragDropEffect	Возвращает список разрешённых операций перетаскивания.
Data	IDataObject	Предоставляет данные из буфера перемещения.
KeyState	Перечисление ***DragDropKeyStates***: AltKey ControlKey RightMouseButton MiddleMouseButton ShiftKey None	Возвращает текущее состояние клавиш SHIFT, CTRL и ALT, а также состояние кнопок мыши.

За управление событием перетаскивания отвечает статический класс **DragDrop**, который грубо говоря управляет буфером.

Сценарий перемещения

В данном примере первый TextBlock является ожидающим данные. Второй TextBlock инициирует событие MouseMove, которое начинает событие перемещение. В обработчике TextBlock_MouseMove происходит доступ к буферу, который заполняется текстом.

```
<UniformGrid Columns="2">
    <TextBlock Text="Привет "
               AllowDrop="True"
               Drop="TextBlock_Drop"
               DragEnter="TextBlock_DragEnter"/>
    <TextBlock Text="Мир!"
               MouseMove="TextBlock_MouseMove"
               Background="Bisque"/>
</UniformGrid>
```

Рисунок 8

Обработчики в связанном классе выглядят так.

```
Ссылок: 1
private void TextBlock_MouseMove(object sender, MouseEventArgs e)
{
    if (e.LeftButton == MouseButtonState.Pressed) //если кнопка зажата
        //Инициируем операцию перетаскивания
        DragDrop.DoDragDrop((TextBlock)sender, ((TextBlock)sender).Text, DragDropEffects.Copy);
        //Этот метод так же изменяет стиль курсора
}
Ссылок: 1
private void TextBlock_DragEnter(object sender, DragEventArgs e)
{
    var s = (TextBlock)sender;
    //Так как событие перемещения инициализировано,
    //при попадании пыши на целевой объект
    //его цвет изменится на бледно-синий
    s.Background = new SolidColorBrush(Colors.LightBlue);
}
Ссылок: 1
private void TextBlock_Drop(object sender, DragEventArgs e)
{
    var s = (TextBlock)sender;
    //Как только клавиша отпущена, получаем текст из буфера
    s.Text += e.Data.GetData(DataFormats.Text);
    //Возвращаем фон в прозрачный
    s.Background = null;
}
```

Рисунок 9

Порядок действий будет таким:

1. Начинает копирование метод DoDragDrop класса DragDrop. Он же сигнализирует систему о том, что начался процесс Drag'N'Drop.
2. После вызывается событие DragEnter в целевом объекте. В примере при попадании мыши в пределы целевого TextBlock, его фон меняется на другой цвет.
3. Пока процесс DragDrop действует, в пределах целевого элемента вызывается событие DragOver (в примере не реализован).
4. Пока процесс DragDrop действует, если мышь покидает границы целевого объекта, он вызывает событие DragLeave (в данном примере не используется).
5. Завершает перемещение событие Drop на целевом объекте. Как только мышь опускается, происходит установка значения из буфера.

Фокус

Фокус - это состояние элемента, при котором он может принимать ввод каких-либо значений. Получить фокус можно разными способами. Например, нажать на `TextBlock` для того, чтобы начать вводить в него текст. Фокус как бы устанавливает цель для ввода. Иногда полезно использовать это свойство для того, чтобы, например,

организовать ввод с цифровой экранной клавиатуры в нужный `TextBlock` из имеющихся.

Название события	Тип маршрутизации	Триггер события
`GotFocus`	Поднимающееся	Получение фокуса
`LostFocus`	Поднимающееся	Потеря фокуса
`GotKeyboardFocus`	Поднимающееся	Получение фокуса с клавиатуры
`PreviewGotKeyboardFocus`	Опускающееся	Получение фокуса с клавиатуры
`LostKeyboardFocus`	Опускающееся	Потеря фокуса с клавиатуры
`PreviewLostKeyboardFocus`	Опускающееся	Потеря фокуса с клавиатуры

У каждого элемента определён ряд событий, связанных с фокусировкой.

Для того, чтобы узнать, находится ли элемент в фокусе, у него есть свойство `Focusable`, в котором true определяет фокус.

```
Focusable="True"
```

Рисунок 10

Области фокуса

Изначально для фокуса определяется одна область, которые принадлежат таким элементам, как `Window`, `Menu`, `ToolBar` и `ContextMenu`.

В этой определённой области только один элемент может быть в «фокусе» - принимать ввод.

Например, можно определить кнопку. И если она в фокусе, специальный класс `FocusManager` сохранит ссылку на него, после чего можно добавлять вводимые клавиши в Content.

```
<Window x:Class="WpfApp1.Window4"
        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:WpfApp1"
        mc:Ignorable="d"
        Title="Window4" Height="450" Width="800"
        KeyDown="Window_KeyDown">
    <Grid>
        <Button Width="150"
                Height="150"/>
    </Grid>
</Window>
```

Рисунок 11

```
private void Window_KeyDown(object sender, KeyEventArgs e)
{
    var b = Keyboard.FocusedElement as Button;
    if (b != null) b.Content += e.Key.ToString();
}
```

Рисунок 12

Рисунок 13

В рамках окна можно определить несколько независимых областей фокуса, где для каждой такой области будет определяться ссылка на элемент фокусировки.

Обычно, его применяют для контейнеров. Можно логически определить области. Для того, чтобы контейнер являлся отдельной областью, у него есть прикрепляемое свойство **`FocusManager.IsFocusScope`**, для которого **`true`** - это согласие на определение отдельной области, false - по умолчанию, не определяет область.

```
<StackPanel FocusManager.IsFocusScope="True"
            x:Name="sp">
  <TextBox Height="20"/>
  <PasswordBox Height="20"/>
</StackPanel>
```

Рисунок 14

Для того, чтобы определить, какой элемент из этой области имеет фокус, необходимо воспользоваться методом **`FocusManager.GetFocusedElement(<элемент>)`**, где элементом является тот самый, в котором определено свойство **`FocusManager.IsFocusScope`** *(поэтому в разметке часто именуют контейнер, который передают в этот метод)*.

```
var a = FocusManager.GetFocusedElement(sp);
```

Рисунок 15

`FocusManager` - это статический класс, который предоставляет набор статических методов, вложенных свойств зависимостей и событий для определения и установки областей фокуса и для установки имеющих фокус элементов в данной области.