# **№ 8 Создание WPF пользовательских ЭУ**

## Задание

1. Разработайте два *пользовательских* элемента управления.
2. Добавьте в них несколько **DependencyProperty** с валидацией через ValidateValueCallback и коррекцией CoerceValueCallback
3. Добавьте события на основе **RoutedEvent**. Используйте разные типы маршрутизации (Direct, Tunneling, Bubbling). Продемонстрируйте разницу между типами маршрутизации.
4. Продемонстрируйте использование элементов в проекте.
5. Создайте пользовательскую команду на основе RoutedUICommand. Используйте ее в проекте.

## Вопросы

1. Что такое Свойства зависимости? Для чего они нужны?
2. Как создатьDependencyProperty?
3. Для чего и как используют делегат ValidateValueCallback?
4. Для чего и как используют делегат CoerceValueCallback?
5. Какие типы маршрутизируемых событий есть в WPF (поясните каждый)?
6. Как создатьRoutedEvent**?**
7. Поясните концепцию Command в WPF? В чем ее преимущества?
8. Как используются команды?
9. Как создать RoutedUICommand?

**1. Свойства зависимости (Dependency Properties)**

**Назначение:** Свойства зависимости позволяют элементам управления автоматически реагировать на изменение данных и поддерживать функционал привязки данных, стилизации, анимации и другие аспекты WPF, связанные с изменяемым состоянием свойств.

**2. Создание DependencyProperty**

Для создания **DependencyProperty** следует выполнить следующие шаги:

1. **Определение Статического Поля:** Определите статическое поле, которое будет хранить само **DependencyProperty**.
2. **Регистрация Свойства:** Используйте метод **DependencyProperty.Register**, чтобы зарегистрировать новое свойство зависимости. Этот метод требует указания имени свойства, типа данных свойства, класса, которому принадлежит свойство, и, опционально, метаданных свойства.

csharpCopy code

public static readonly DependencyProperty IsSpinningProperty = DependencyProperty.Register( "IsSpinning", typeof(bool), typeof(MyControl), new PropertyMetadata(false));

**3. Делегат ValidateValueCallback**

**Назначение:** **ValidateValueCallback** используется для проверки значения свойства зависимости перед его установкой. Этот делегат может предотвратить установку недопустимых значений.

**Пример использования:**

csharpCopy code

private static bool ValidateValue(object value) { int val = (int)value; return val >= 0 && val <= 100; // Пример: значение должно быть от 0 до 100 } public static readonly DependencyProperty MyProperty = DependencyProperty.Register( "My", typeof(int), typeof(MyClass), new PropertyMetadata(0, null, null, ValidateValue));

**4. Делегат CoerceValueCallback**

**Назначение:** **CoerceValueCallback** применяется для коррекции или изменения значения свойства зависимости после его валидации, но перед установкой. Это может быть использовано для приведения значения к допустимому диапазону или зависимости от других свойств.

**Пример использования:**

csharpCopy code

private static object CoerceValue(DependencyObject d, object baseValue) { int value = (int)baseValue; if (value > 100) value = 100; // Ограничить значение сверху значением 100 return value; } public static readonly DependencyProperty MyProperty = DependencyProperty.Register( "My", typeof(int), typeof(MyClass), new PropertyMetadata(0, null, CoerceValue));

**5. Типы маршрутизируемых событий в WPF**

1. **Direct:** Событие вызывается и обрабатывается на объекте, где оно было возбуждено, без вовлечения других элементов.
2. **Bubbling:** Событие возникает на дочернем элементе и "всплывает" к родительским элементам до корня визуального дерева.
3. **Tunneling:** Событие начинается от корня и "проходит" к целевому элементу, вызываясь последовательно на каждом элементе по пути.

**6. Создание RoutedEvent**

**Процесс создания:**

csharpCopy code

public static readonly RoutedEvent TapEvent = EventManager.RegisterRoutedEvent( "Tap", RoutingStrategy.Bubble, typeof(RoutedEventHandler), typeof(MyCustomControl)); // Добавление CLR обработчика public event RoutedEventHandler Tap { add { AddHandler(TapEvent, value); } remove { RemoveHandler(TapEvent, value); } }

**7. Концепция Command в WPF**

**Назначение:** Команды в WPF используются для инкапсуляции логики действий, что позволяет централизованно управлять доступностью и активностью функций интерфейса.

**Преимущества:**

* **Повышение Переиспользуемости:** Логика действий может быть переиспользована в разных частях приложения.
* **Разделение Логики:** Команды позволяют разделить логику управления и интерфейс пользователя.

**8. Использование Команд**

Команды могут быть привязаны к элементам интерфейса, таким как кнопки или меню, и автоматически управлять их доступностью на основе состояния команды.

**9. Создание RoutedUICommand**

**Процесс создания:**

csharpCopy code

public static readonly RoutedUICommand ExitCommand = new RoutedUICommand( "Exit", "Exit", typeof(MyCommands), new InputGestureCollection { new KeyGesture(Key.F4, ModifierKeys.Alt) });

Каждый аспект в этих ответах углубляет понимание инструментов разработки, доступных в WPF, что способствует созданию более функциональных и адаптируемых приложений.