Команды

Основы команд

В WPF кроме обработки событий приложение может взаимодействовать с пользователем с помощью команд. **Команды** представляют механизм выполнения какой-нибудь задачи, например, копирования текста - когда мы нажимаем Ctrl+C, то мы копируем текст в буффер. В процессе копирования выполняется ряд действий, и все вместе эти действия объединяются в одну команду. Использование команд помогает нам сократить объем кода и использовать одну и ту же команду для нескольких элементов управления в различных местах программы. Таким образом, команды позволяют абстрагировать набор действий от конкретных событий конкретных элементов.

В некотором роде команды в WPF являются реализацией общераспространенного паттерна Команда.

Модель команд в WPF состоит из четырех аспектов:

- Сама команда, которая представляем выполняемую задачу
- Привязка команд, которая связывает команду с определенной логикой приложения
- Источник команды - элемент пользовательского интерфейса, который запускает команду (например, кнопка, по нажатию который выполняется команда)
- Цель команды - элемент интерфейса, на котором выполняется команда

Команда

Все команды реализуют интерфейс System.Window.Input.ICommand:

```
public interface ICommand
{
    event EventHandler CanExecuteChanged;
    void Execute (object parameter);
    bool CanExecute (object parameter);
}
```

Метод **Execute** предназначен для хранения логики команды. Функция **CanExecute** возвращает true, если команда включена и доступна для использования, и false, если команда отключена. Событие **CanExecuteChanged** вызывается при изменении состояния команды.

В WPF этот интерфейс реализован встроенным классом **System.Windows.Input.RoutedCommand**, который является базовым для всех встроенных команд. Поэтому, если нам потребуется создать свой класс команды, мы можем либо реализовать ICommand, либо унаследовать свой класс команды от RoutedCommand.

WPF уже обладает большим набором встроенных команд. Все они представляют объекты класса **RoutedUICommand**, который является производным от RoutedCommand.

Все встроенные команды объединяются в семь групп, и каждую такую группу представляет отдельный статический класс. А отдельные команды реализованы как статические свойства этих классов:

Общие команды приложения представлены классом **ApplicationCommands**. Это команды: **CancelPrint** (Отменить печать), **Close** (Закрыть), **ContextMenu** (Конекстное меню), **Copy** (Копировать), **CorrectionList** (Список исправлений), **Cut** (Вырезать), **Delete** (Удалить), **Find** (Найти), **Help** (Справка), **New** (Создать), **Open** (Открыть), **Paste** (Вставить), **Prit** (Печать), **PrintPreview** (Предварительный просмотр), **Properties** (Свойства), **Redo** (Повторить), **Replace** (Заменить), **Save** (Сохранить), **SaveAs** (Сохранить как), **SelectAll** (Выделить все), **Stop** (Остановить), **Undo** (Отменить) и т.д.

Команды навигации применяются для навигации по содержимому, например, в браузерных приложениях. Они представлены классом **NavigationCommands**: **BrowseBack** (Назад), **BrowseForward** (Вперед), **BrowseHome** (Домой / На главную страницу)

, **BrowseStop** (Остановить), **Favorites** (Избранное), **FirstPage** (Первая страница), **GoToPage** (Переход), **LastPage** (Последняя страница), **NextPage** (Следующая страница), **PreviousPage** (Предыдущая страница), **Refresh** (Обновить) и т.д.
Команды компонентов интерфейса используются для перемещения и выделения содержимого элементов управления. Они представлены классом **ComponentCommands**: **MoveDown** (Переместить курсор вниз), **MoveLeft** (Переместить курсор влево), **MoveRight** (Переместить курсор вправо), **MoveUp** (Переместить курсор вверх), **ScrollPageDown** (Прокрутить вниз), **SelectToEnd** (Прокрутить вверх) и т.д.

Команды редактирования документов представлены классом **EditingCommands**: **AllignCenter** (Выравнивание по центру), **DecreaseFontSize** (Уменьшение высоты шрифта), **MoveDownByLine** (Переход на строку вниз) и т.д.

Команды для управления мультимедиа представлены классом **MediaCommands**: **DecreaseVolume** (Уменьшить громкость), **Play** (Воспроизвести), **Rewind** (Перемотка), **Record** (Запись)

Системные команды представлены классом **SystemCommands**: **CloseWindow** (Закрыть окно приложения), **MaximizeWindow** (Развернуть окно), **MinimizeWindow** (Свернуть окно), **RestoreWindow** (Восстановить окно) и т.д.

Команды ленты панели инструментов представлены классом **RibbonCommands**: **AddToQuickAccessToolBar** (Добавить на для быстрого доступа), **MaximizeRibbonCommand** (Развернуть ленту панели инструментов) и т.д.

Это только некоторые команды. И гораздо много, и они охватывают множество ситуаций.

Источник команд

Источником команды является элемент, который вызывает команду. Однако не каждый элемент управления может быть источником. Для этого он должен реализовать интерфейс ICommandSource:

```
public interface ICommandSource
{
    ICommand Command {get;}
    object CommandParameter {get;}
    IInputElement CommandTarget {get;}
}
```

- Свойство Command представляет выполняемую команду. Однако это свойство игнорируется системой, если объект, реализующий интерфейс ICommand, не является наследником класса RoutedCommand.

Как понятно из объявления интерфейса, элемент, его реализующий, может принимать только одну команду.

- Свойство CommandParameter представляет параметр выполняемой команды - это те данные, которые передаются команде
- Свойство CommandTarget представляет цель команды, то есть элемент, для которого выполняется команда. Нередко в качестве цели команды выступает тот же самый элемент, который и вызывает команду, а свойство имеет значение null

Допустим, если у нас есть кнопка, и мы хотим к ней добавить команду ApplicationCommands.Help, то мы могли бы в коде XAML прописать так:

```
<Button x:Name="helpButton" Command="ApplicationCommands.Help" Content="Help" />
```

Также допустимо сокращение название команды:

```
<Button x:Name="helpButton" Command="Help" Content="Help" />
```

Либо можно сделать это в коде C#:

```
helpButton.Command = ApplicationCommands.Help;
```

Привязка команд

Все команды, в том числе и встроенные, не содержат конкретного кода по их выполнению. Это просто специальные объекты, которые представляют некоторую задачу. Чтобы связать эти команды с реальным кодом, который бы выполнял некоторые действия, нужно использовать привязку команд.

Привязка команд представляет объект **CommandBinding**. Его событие Executed прикрепляет обработчик, который будет выполняться при вызове команды. А свойство Command уставливает саму команду, к которой относится обработчик.

Обычная форма установки привязки команды, например, где-нибудь в конструкторе класса MainWindow:

```
// создаем привязку команды
CommandBinding commandBinding = new CommandBinding();
// устанавливаем команду
commandBinding.Command = ApplicationCommands.Help;
// устанавливаем метод, который будет выполняться при вызове команды
commandBinding.Executed += CommandBinding_Executed;
// добавляем привязку к коллекции привязок элемента Button
helpButton.CommandBindings.Add(commandBinding);
```

Команда добавляется в коллекцию CommandBindings элемента Button. И после этого, если мы вызовем команду, то будет выполняться метод CommandBinding_Executed, который может быть определен, к примеру, следующим образом:

```
private void CommandBinding_Executed(object sender, ExecutedRoutedEventArgs e)
{
    MessageBox.Show("Справка по приложению");
}
```

Также мы можем определить привязку в коде XAML:

```
<Button x:Name="helpButton" Command="ApplicationCommands.Help" Content="Help">
    <Button.CommandBindings>
        <CommandBinding Command="Help" Executed="CommandBinding_Executed" />
    </Button.CommandBindings>
</Button>
```

Это форма определения привязки команды будет аналогично той, что использовалась в коде C#.

Маршрутизация команд

Как и события, команды в WPF являются маршрутизированными. А это значит, что команду можно вызвать на одном элементе и она будет идет вверх от вложенного элемента к контейнеру. К примеру, определим следующий код xaml:

```
<Window x:Class="CommandsApp.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:CommandsApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="300">
    <Window.CommandBindings>
        <CommandBinding Command="Help" Executed="WindowBinding_Executed" />
    </Window.CommandBindings>
    <Grid>
        <Button x:Name="helpButton" Command="ApplicationCommands.Help" Content="Help" />
    </Grid>
</Window>
```

И в файле кода пропишем метод WindowBinding_Executed:

```
private void WindowBinding_Executed(object sender, ExecutedRoutedEventArgs e)
{
    MessageBox.Show("Вызов справки");
}
```

При нажатии на кнопку команда пойдет вверх от кнопки, которая ее вызвала, к объектам контейнерам - Grid и Window. И так как у элемента Window в коллекцию привязок добавлена привязка для команды Help, то она будет использоваться для выполнения этой команды.

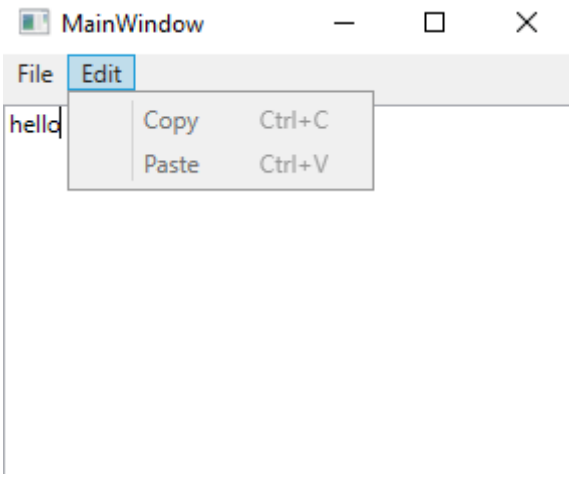
Пример использования команд

Теперь объединим все и создадим систему для вызова команды. Для этого определим код XAML:

```
<Window x:Class="CommandsApp.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:CommandsApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="300">
    <DockPanel>
        <Menu DockPanel.Dock="Top" MinHeight="25">
            <MenuItem Header="File" />
            <MenuItem Header="Edit">
                <MenuItem Header="Copy" CommandTarget="{Binding ElementName=txtBox}" Command="Cc
                <MenuItem Header="Paste" CommandTarget="{Binding ElementName=txtBox}" Command="P
            </MenuItem>
        </Menu>
        <TextBox x:Name="txtBox" />
    </DockPanel>
</Window>
```

Здесь два пункта меню Copy и Paste будут вызывать соответствующие команды. Причем здесь мы не задаем привязку для этих команд, так как для команд Copy, Cut, Redo, Undo и Paste уже опредлены встроенные привязки. Поэтому в данном случае нам не надо вносить в файл кода C# никаких изменений.

Кроме того, здесь с помощью выражения CommandTarget="{Binding ElementName=txtBox}" мы указываем, что команды будут направлены на текстовое поле, которое будет целью команд. И теперь запустим проект:



Если в буфере обмена ничего нет, то команда Paste будет неактивной. Также если в текстовом поле не выделен текст, то пункт меню Copy также будет не активным. При выделении текста становится активным пункт меню Copy, а после копирования в буфер и пункт меню Paste.

Создание новых команд

В WPF определено много команд, но даже их может оказаться недостаточно. Поэтому нередко разработчики создают свои собственные команды. Наиболее простой способ создания команды - использование готовых классов **RoutedCommand** и **RoutedUICommand**, в которых уже реализован интерфейс ICommand. Итак, в файле кода под классом окна создадим новый класс, назовем его WindowCommands, который будет содержать новые команды:

```
using System.Windows.Input;

public class WindowCommands
{
    static WindowCommands()
    {
        Exit = new RoutedCommand("Exit", typeof(MainWindow));
    }
    public static RoutedCommand Exit { get; set; }
}
```

В данном случае команда называется Exit и представляет собой объект RoutedCommand. В конструктор этого объекта в данном случае передается название команды и элемент-владелец команды (здесь объект MainWindow). Причем команда определяется как статическое свойство.

Теперь в коде XAML установим привязку к этой команде и задействуем ее:

```
<Window x:Class="CommandsApp.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:CommandsApp"
        mc:Ignorable="d"
        Title="MainWindow" Height="250" Width="300">
    <Window.CommandBindings>
        <CommandBinding Command="local:WindowCommands.Exit" Executed="Exit_Executed"/>
    </Window.CommandBindings>
    <Grid>
        <Menu VerticalAlignment="Top" MinHeight="25">
            <MenuItem Header="Выход" Command="local:WindowCommands.Exit"  />
        </Menu>
        <Button x:Name="Button1" Width="80" Height="30" Content="Выход"
                Command="local:WindowCommands.Exit"  />
    </Grid>
</Window>
```

Итак, здесь есть пункт меню и есть кнопка, которые вызывают эту команду. Так как команды определены в локальном пространстве имен, которое соответствует префиксу local и представляют статические свойства, то к ним обращаемся с помощью выражения local:WindowCommands.Exit. Привязка команды связывает ее с обработчиком Exit_Executed, который определим в коде c#:

```
private void Exit_Executed(object sender, ExecutedRoutedEventArgs e)
{
    using (System.IO.StreamWriter writer = new System.IO.StreamWriter("log.txt", true))
    {
        writer.WriteLine("Выход из приложения: " + DateTime.Now.ToShortDateString() + " " +
        DateTime.Now.ToLongTimeString());
        writer.Flush();
    }

    this.Close();
}
```

Таким образом, при нажатии на кнопку или пункт меню будет выполняться команда Exit, которая будет вызывать вышеопределенный обработчик. В нем происходит запись в лог о дате и времени выхода и собственно осуществляется выход из приложения.