**Справка по C#.**

Оглавление.

[Создание и обращение к новой форме. 1](#_Toc507279632)

[Перевести число из 10-ой с. с. в другую. 1](#_Toc507279633)

[Создать динамический массив и изменить кол-во элементов в нём. 2](#_Toc507279634)

[Полезные сторонние библиотеки. 2](#_Toc507279635)

[Работа с диалогами. 2](#_Toc507279636)

[Анимация с использованием таймера. 3](#_Toc507279637)

[Обработка нажатий на клавиши. 4](#_Toc507279638)

[Убрать выделение элемента при запуске программы и как перевести фокус на элемент. 4](#_Toc507279639)

[Текущее время компьютера. 5](#_Toc507279640)

[try catch. 5](#_Toc507279641)

[Работа с 3D графикой в WPF. 5](#_Toc507279642)

[Привязка одних значений к другим. 9](#_Toc507279643)

[Координаты курсора в WPF. 9](#_Toc507279644)

[Реализация интерфейса INotifyPropertyChanged. 10](#_Toc507279645)

[Реализация интерфейса DependencyObject. 11](#_Toc507279646)

[Проверка состояния нажатия кнопки мыши в WPF. 11](#_Toc507279647)

[Добавление класса Bitmap в WPF Проект. 12](#_Toc507279648)

# Создание и обращение к новой форме.

1. Создаём новую форму в проекте: Project -> Add Windows Form…
2. В обработчике события добавляем код:

Form3 NForm = new Form3();

NForm.ShowDialog();

# Перевести число из 10-ой с. с. в другую.

Convert.ToString(InNum, Sys) == Строка в которой содержится InNum в Sys

системе счисления при том, что Sys может быть или 2, или 8, или 16.

# Создать динамический массив и изменить кол-во элементов в нём.

int[] Mas = new int [N]; //N – натуральное число или 0, кол-во элементов

Array.Resize<int>(ref Mas, NewN); //NewN – новое кол-во элементов

# Полезные сторонние библиотеки.

Math.NET //Дополнительная математическая библиотека

# Работа с диалогами.

if (dialogName.ShowDialog() == DialogResult.OK)

{ } //Вызвать диалог и выполнить условие, если результат диалога OK

DialogResult = DialogResult.OK; //Если вызывается диалог с другой формой

# Анимация с использованием таймера.

Timer tmr = new Timer(); //Win. Forms

DispatcherTimer tmr = new DispatcherTimer(); //WPF

tmr.Interval == значение интервала в миллисекундах

tmr.Tick += new EventHandler((o, ev) =>

{ …Что должно произойти… });

Или так:

tmr.Tick += new EventHandler(TimerEventProcessor);

void TimerEventProcessor(Object o, EventArgs ev)

{ …Что должно произойти… }

tmr.Start(); //Запустить таймер

tmr.Stop(); //Остановить таймер

# Обработка нажатий на клавиши.

Для мыши.

В обработчике события должна существовать MouseEventArgs e.

Подходит событие MouseDown.

if (e.Button == MouseButtons.Left)

{ …Что должно произойти… }

Для клавиатуры.

Пишем в обработчике события KeyDown.

if (e.KeyCode == Keys.L); //Вместо L почти любая кнопка

Такие кнопки как TAB, RETURN, ESC и стрелочки могут быть использованы для обработки события нажатия на них только, если на форме свойство KeyPreview принимает значение True. Иногда в событии PreviewKeyDown должно быть написано:

e.IsInputKey = true;

# Убрать выделение элемента при запуске программы и как перевести фокус на элемент.

elementName.TabStop = false;

Элемент с TabIndex == 0 выделяется при запуске программы.

elementName.Focus(); //Перевести фокус на нужный элемент

# Текущее время компьютера.

DateTime.Now.Millisecond, DateTime.Now.Second и т.п.

# try catch.

try { body.Margin = new Thickness(67, 497 - Y0 \* 100 \* scale, 0, 0); }

catch (Exception) {}

# Работа с 3D графикой в WPF.

Треугольник.

Создадим камеру и добавим её на Viewport3D:

PerspectiveCamera perCam = new PerspectiveCamera();

perCam.Position = new Point3D(0, 0, 5);

perCam.LookDirection = new Vector3D(0, 0, -1);

perCam.UpDirection = new Vector3D(0, 1, 0);

perCam.FieldOfView = 60;

vp3d.Camera = perCam;

Создадим освещение и добавим его на Viewport3D:

DirectionalLight dLight = new DirectionalLight();

dLight.Color = Colors.White;

dLight.Direction = new Vector3D(0, 0, -1);

ModelVisual3D mv3d = new ModelVisual3D();

Model3DGroup m3dg = new Model3DGroup();

m3dg.Children.Add(dLight);

//m3dg.Children.Add(другой источник света);

mv3d.Content = m3dg;

vp3d.Children.Add(mv3d);

Далее по схеме. В белой области пример связи нижнего с тем, что выше.

Создадим точки:

Point3D point0 = new Point3D();

point.X = 1;

point.Y = 1;

point.Z = 1;

Таким образом создаём ещё 2 точки.

Создадим сетку треугольника(ов):

MeshGeometry3D mesh = new MeshGeometry3D();

mesh.Positions.Add(point0);

mesh.Positions.Add(p1);

mesh.Positions.Add(p2);

mesh.TriangleIndices.Add(0);

mesh.TriangleIndices.Add(2);

mesh.TriangleIndices.Add(1);

Нужен именно такой порядок добавления, если хотим сделать треугольник видимым в положительном направлении оси Z. В одну сетку (Mesh) можно объединить несколько треугольников.

Создадим материал поверхности:

SolidColorBrush brush = new SolidColorBrush(Colors.Red);

DiffuseMaterial difMaterial = new DiffuseMaterial(brush);

MaterialGroup materialGroup = new MaterialGroup();

materialGroup.Children.Add(difMaterial);

//materialGroup.Children.Add(другой материал);

Соберём всё в модель:

GeometryModel3D geometry = new GeometryModel3D(mesh, materialGroup);

ModelUIElement3D model = new ModelUIElement3D();

model.Model = geometry;

В конце добавим получившуюся модель на Viewport3D:

vp3d.Children.Add(model);

Можно было добавить камеру и свет другим способом:

В XAML файл пишем код по следующему образцу:

<Viewport3D x:Name="vp3d" HorizontalAlignment="Left" Height="570" VerticalAlignment="Top" Width="992">

<Viewport3D.Camera>

<PerspectiveCamera

FarPlaneDistance="100"

LookDirection="-11,-10,-9"

UpDirection="0,1,0"

NearPlaneDistance="1"

Position="11,9,9"

FieldOfView="70"/>

</Viewport3D.Camera>

<ModelVisual3D>

<ModelVisual3D.Content>

<DirectionalLight

Color="White"

Direction="-2,-3,-1"/>

</ModelVisual3D.Content>

</ModelVisual3D>

</Viewport3D>

# Привязка одних значений к другим.

Привязка угла поворота 3D модели к значению слайдера.

AxisAngleRotation3D axisAngRot3d = new AxisAngleRotation3D(new Vector3D(0, 1, 0), 0);

RotateTransform3D qRotation = new RotateTransform3D(axisAngRot3d);

BindingOperations.SetBinding(axisAngRot3d, AxisAngleRotation3D.AngleProperty, new Binding("Value") { Source = slider });

…

geometry.Transform = qRotation;

# Координаты курсора в WPF.

Относительно некоторого элемента.

Point XYCoor = Mouse.GetPosition(Относ-но какого элем-а на форме);

Относительно экрана монитора.

Point XYScreenCoor = (Какой-нибудь элемент).PointToScreen(Mouse.GetPosition(Тот же самый элемент));

# Реализация интерфейса INotifyPropertyChanged.

Может быть применено для связи значения «переменной» и св-ва элемента, но только в сторону элемента, т.е. изменение значения св-ва элменета НЕ будет влиять на значение «переменной».

class Author : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

private string foreName;

…

public string ForeName

{

get { return foreName; }

set

{

foreName = value;

OnPropertyChanged(“ForeName”);

}

}

…

void OnPropertyChanged(string propName)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propName));

// То же самое, что и

// If(PropertyChanged != null) PropertyChanged(this, new

// PropertyChangedEventArgs(propName));

}

}

Author author;

Затем можно связать, например, author.ForeName и textbox.Text.

# Реализация интерфейса DependencyObject.

Может быть применено для связи значений в обе стороны.

public class Phone : DependencyObject

{

public static readonly DependencyProperty TitleProperty =

DependencyProperty.Register("Title", typeof(string), typeof(Phone));

public string Title

{

get { return (string)GetValue(TitleProperty); }

set { SetValue(TitleProperty, value); }

}

}

# Проверка состояния нажатия кнопки мыши в WPF.

If (Mouse.LeftButton == MouseButtonState.Pressed) { }

# Добавление класса Bitmap в WPF Проект.

Нужно добавить ссылку на сборку через References и подключить пространство имён: using System.Drawing.Imaging;