Работа с жестами

В процессе работы с мобильными программными приложениями, запущенных на мобильных устройствах с сенсорными экранами, пользователи чаще всего взаимодействуют с элементами управления пользовательских интерфейсов с помощью жестов. Поэтому необходимо изучить порядок работы с жестами Tap (касание), Pinch (сжатие), Pan (сдвиг), Swip (прокрутка), а также перетаскивания и отпускания (Drag-and-Drop), которые поддерживаются подплатформой Xamarin.Forms с использованием класса GestureRecognizer.

Сначала рассматриваются особенности работы с жестами касания (Tap). Чтобы сделать элемент управления на пользовательском интерфейсе мобильного программного приложения доступным для жеста касания, необходимо:

1. Создать экземпляр распознавателя TapGestureRecognizer, обрабатывающий событие Tapped.
2. Добавить новый распознаватель жестов в коллекцию GestureRecognizers для элемента управления пользовательского интерфейса мобильного программного приложения.

В приведенном ниже фрагменте программного кода Си Шарп приведен пример создания распознавателя жеста TapGestureRecognizer, присоединенного к элементу Image.

```
var tapGestureRecognizer =
                    new TapGestureRecognizer();
tapGestureRecognizer.Tapped += (s, e) =>
{
   // Программный код для обработки касания
};
image.GestureRecognizers.Add(tapGestureRecognizer);
```

По умолчанию элемент управленя Image будет реагировать на одиночные касания. Для реагирования на двойные (или множественные) касания производится задание значения свойства NumberOfTapsRequired.

Распознаватель жестов можно добавить к элементу управления пользовательского интерфейса мобильного программного приложения с помощью программного кода XAML с помощью присоединенных свойств. Далее приведен фрагмент программного кода XAML для распознавателя двойного касания TapGestureRecognizer к элементу управления Image. Программный код производит обращение к обработчику OnTapGestureRecognizerTapped события Tapped, которое возникает при жесте двойного касания по картинке, отображаемой в элементе управления Image. Программный код обработчика содержится в файле программной части с расширением xaml.cs.

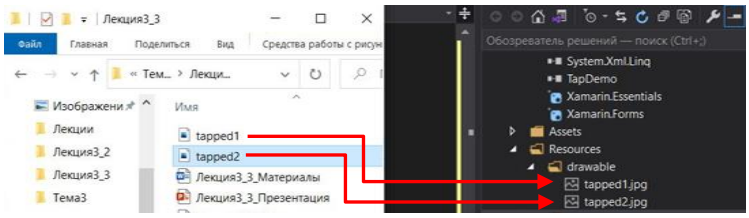
```
<Image Source="tapped.jpg">
   <Image.GestureRecognizers>
      <TapGestureRecognizer
           Tapped="OnTapGestureRecognizerTapped"
```

```
            NumberOfTapsRequired="3" />
    </Image.GestureRecognizers>
</Image>
```

Для демонстрации возможностей мобильного программного приложения Xamarin.Forms по работе с жестами касания создается проект TapDemo для мультиплатформенного приложения Xamarin.Forms с использованием шаблона «Пустой». После создания проекта необходимо зайти в диалоговое окно «Обозреватель решений», открыть один из трех созданных проектов TapDemo (без суффикса), выбрать файл MainPage.xaml и удалить его из проекта. При этом будет удален и сопутствующий файл MainPage.xaml.cs.

Далее с использованием рассмотренной ранее последовательности действий необходимо в проекте TapDemo создать файл MainPage.cs для класса MainPage. В файле MainPage.cs с помощью программного кода на языке Си Шарп будет задаваться внешний вид пользовательского интерфейса программного приложения.

Далее необходимо в диалоговом окне «Обозреватель решений» выделить и раскрыть проект TapDemo.Android, открыть папку Resources и в папку Drawable этого проекта перенести файлы изображений tapped1.jpg и tapped2.jpg из внешнего источника. При этом необходимо учесть, что в файле tapped1.jpg хранится изображение в цветном фрмате, а в файле tapped2.jpg хранится то же изображение, но в черно-белом формате.



Далее необходимо открыть файл MainPage.cs и набрать приведенный ниже программный код Си Шарп.

```
using System;
using Xamarin.Forms;
using System.Diagnostics;

namespace TapDemo
{
    public class MainPage: ContentPage
     {
            int tapCount;
            readonly Label lbl;

            public MainPage()
            {
// Создание элемента управления Image
// и задание значений его свойств
                    var image = new Image
```

```
                    {
                        Source = "tapped1.jpg"
                        HorizontalOptions =
                                        LayoutOptions.Center,
                        VerticalOptions =
                                  LayoutOptions.CenterAndExpand,
                         Aspect = Aspect.Fill,
                     };

// Создание экземпляра tapGestureRecognizer
                  var tapGestureRecognizer =
                                       new TapGestureRecognizer();

// Задание значения свойства NumberOfTapsRequired
// для двух касаний
                tapGestureRecognizer.NumberOfTapsRequired = 2;

// Обработка жеста касания с помощью
// процедуры OnTapGestureRecognizerTapped

                 tapGestureRecognizer.Tapped +=
                                    OnTapGestureRecognizerTapped;

// Добавление обработчика события к
// элементу управления Image
           image.GestureRecognizers.Add(tapGestureRecognizer);
//Создание элемента управления Label и
// задание значений его свойств
                     lbl = new Label
                     {
                    Text =
                      "Сделайте двойное касание по картинке!",
                    TextColor = Color.Black,
                    FontAttributes=FontAttributes.Bold,
                    HorizontalTextAlignment =
                                             TextAlignment.Center,
                    VerticalTextAlignment = TextAlignment.Center,
                    FontSize =
              Device.GetNamedSize
                       (NamedSize.Large, typeof(Label)),
                    HorizontalOptions = LayoutOptions.Center,
                    VerticalOptions =
                     LayoutOptions.CenterAndExpand
                     };
//Создание макета StackLayout и включение его в страницу
```

```
                    BackgroundColor = Color.Aqua;
                    Content = new StackLayout
                    {
                        Padding = new Thickness(20, 100),
                        Children =
                            {image, lbl}
                     };
            }
// Процедура OnTapGestureRecognizerTapped –
// обработчик касания
void OnTapGestureRecognizerTapped
                            (object sender, EventArgs args)
              {
// Увеличение значения количества касаний
                tapCount++;
// Отображение количества касаний
// в свойстве Text элемента lbl
lbl.Text =
          String.Format("{0}{1} двойное нажатие произведено!",
                tapCount,
                tapCount == 0 ? "" : "-e");
                var imageSender = (Image)sender;
// Перевод изображения из цветного в черно-белое и обратно
                 if (tapCount % 2 == 0)
                    {
                        imageSender.Source = "tapped1.jpg";
                     }
                     else
                     {
                         imageSender.Source = "tapped2.jpg";
                     }
              }
        }
}
```

В программном коде в странице типа ContentPage с помощью свойства Content помещается макет StackLout, внутри которого, в свою очередь, с помощью свойства Children располагаются элементы управления Image с именем image и Label с именем lbl. Задаются значения свойств элемента управления Label, которые определяют содержание текста внутри элемента управления, цвет фона, стиль шрифта, размещение текста по горизонтали и вертикали внутри элемента управления, а также вертикальное и горизонтальное размещение элемента управления внутри макета. Размер шрифта в элементе управления Label адаптирован к типу мобильного устройства, на котором запускается мобильное программное приложение.

Задаются значения свойств элемента управления Image, которые определяют источник изображения, стиль заполнения изображением элемента управления, а также вертикальное и горизонтальное размещение элемента управления внутри макета.

Производится создание экземпляра класса TapGestureRecognizer, а также задание значения свойства NumberOfTapsRequired для двух касаний.

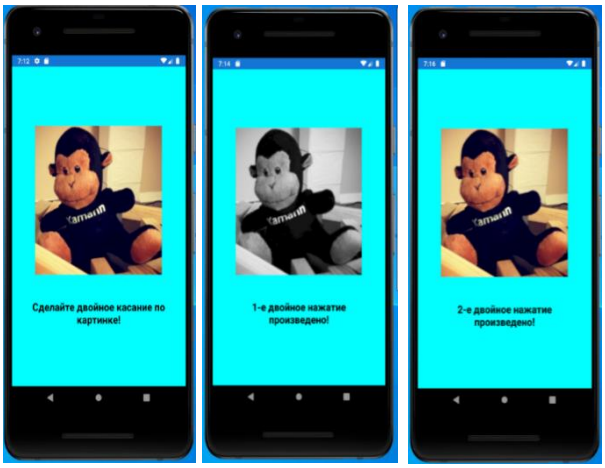
Жест касания tapGestureRecognizer добавляется к элементу управления Image с использованием GestureRecognizers. Обработка жеста касания производится с помощью обработчика OnTapGestureRecognizerTapped.

В программном коде обработчика OnTapGestureRecognizerTapped после двойного касания по элементу управления Image увеличивается значение переменной tapCount. Затем с помощью условного оператора реализуется проверка условия с помощью переменной tapCount. После каждого двойного касания по элементу управления Image происходит смена картинки, отображаемой в элементе управления Image. При четном значении переменной tapCount производится загрузка цветной картинки из файла с именем tapped1.jpg. При нечетном значении переменной tapCount производится загрузка черно-белой картинки из файла с именем tapped2.jpg.

Также в случае двойного касания по элементу управления Image в элементе управления Label с именем lbl отображается сообщение о количестве касаний.

При этом обработчик событий выполняется только в случае, если два касания были совершены в течение определенного периода времени. Если второе касание не было совершено в течение этого промежутка времени, оно игнорируются, а счетчик касаний tapCount не изменяет значение.

Для демонстрации возможности работы мобильного программного приложения с жестом касания необходимо сначала собрать программное приложение и, в случае отсутствия ошибок, запустить его. При запуске программного приложения в пользовательском интерфейсе отображается цветная картинка с расположенной под ней поясняющим текстом о необходимости произвести двойное касание по картинке. При каждом двойном касании к картинке в отображаемом под ним тексте выдается сообщение о количестве сделанных двойных касаний. При этом, если второе касание не было совершено в течение необходимого промежутка времени, то количество сделанных двойных касаний не изменяется.



Далее рассматриваются особенности работы мобильных программных приложений с жестами сжатия с помощью класса PinchGestureRecognizer. Жест сжатия предназначен для интерактивного изменения размера изображения внутри элемента управления.

Чтобы сделать доступным элемент управления пользовательского интерфейса для жеста сжатия, необходимо:

1. Создать экземпляр класса PinchGestureRecognizer.
2. Реализовать обработку события PinchUpdated и добавить распознаватель жестов в коллекцию класса GestureRecognizers для элемента управления.

В приведенном ниже фрагменте программного кода Си Шарп приводится пример реализации распознавания жеста сжатия для элемента управления Image за счет использования распознавателя PinchGestureRecognizer.

```
var pinchGesture = new PinchGestureRecognizer();
pinchGesture.PinchUpdated += (s, e) => {
  // Программный код для обработки жеста сжатия
};
image.GestureRecognizers.Add(pinchGesture);
```

Также распознавание жеста сжатия элемента управления Image, в котором отображается картинка, с помощью класса PinchGestureRecognizer можно реализовать в программном коде XAML, фрагмент которого приведен ниже. Программный код производит обращение к обработчику OnPinchUpdated события PinchUpdated, возникающего в случае изменения размеров картинки внутри элемента управления Image. Программный код обработчика должен находиться в файле с расширением xaml.cs.

```
<Image Source="waterfront.jpg">
  <Image.GestureRecognizers>
    <PinchGestureRecognizer
            PinchUpdated="OnPinchUpdated" />
  </Image.GestureRecognizers>
</Image>
```

Для выполнения преобразований в пользовательском интерфейсе, которые должны быть выполнены в ответ на жест сжатия, требуется

произвести ряд операций по масштабированию элемента управления, на который воздействует жест сжатия. Для этого обычно создается универсальный вспомогательный класс для выполнения интерактивного масштабирования элемента управления. Далее приводится фрагмент программного кода Си Шарп для универсального вспомогательного класса с именем PinchToZoomContainer.

```
public class PinchToZoomContainer : ContentView
{
 ...
 public PinchToZoomContainer ()
 {
  var pinchGesture = new PinchGestureRecognizer ();
  pinchGesture.PinchUpdated += OnPinchUpdated;
  GestureRecognizers.Add (pinchGesture);
 }
 void OnPinchUpdated (object sender,
                      PinchGestureUpdatedEventArgs e)
 {
  // Программный код для обработки жеста сжатия
 }
}
```

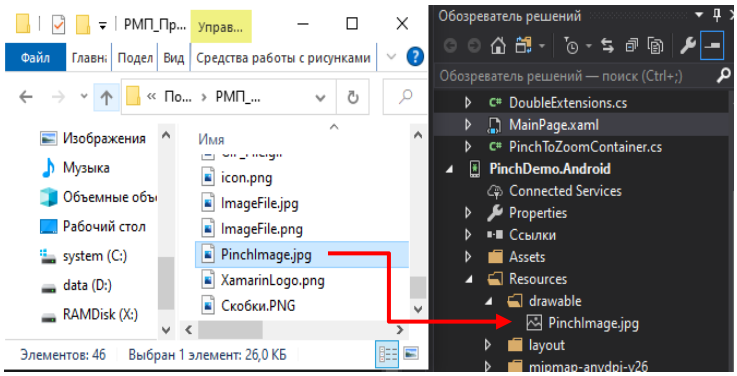
Универсальный вспомогательный класс может служить оболочкой для элемента управления, внутри которого содержимое масштабируется в ответ на жест сжатия.

Далее приведен фрагмент программного кода XAML, в котором класс PinchToZoomContainer служит оболочкой для элемента управления Image.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        xmlns:local=
           "clr-namespace:PinchGesture;assembly=PinchGesture"
         x:Class="PinchGesture.HomePage">
  <ContentPage.Content>
    <Grid Padding="20">
      <local:PinchToZoomContainer>
        <local:PinchToZoomContainer.Content>
          <Image Source="PinchImage.jpg" />
        </local:PinchToZoomContainer.Content>
      </local:PinchToZoomContainer>
    </Grid>
  </ContentPage.Content>
</ContentPage>
```

Для демонстрации возможностей по работе мобильного программного приложения жестом сжатия создается проект PinchDemo для мультиплатформенного приложения Xamarin.Forms с использованием шаблона «Пустой». После создания проекта необходимо зайти в диалоговое

окно «Обозреватель решений» выделить проект PinchDemo и с помощью рассмотренной ранее последовательности действий создать файлы классов DoubleExtensions и PinchToZoomContainer. Далее в диалоговом окне «Обозреватель решений» необходимо выделить и раскрыть проект PinchDemo.Android открыть папку Resources и перенести файл с именем PinchImage.jpg из внешнего источника в папку Drawable .



Далее необходимо в диалоговом окне «Обозреватель решений» выделить и открыть файл DoubleExtensions.cs и далее набрать приведенный ниже программный код Си Шарп.

```
using System;
namespace PinchDemo
{
  public static class DoubleExtensions
  {
      public static double Clamp
                  (this double self, double min, double max)
          {
              return Math.Min (max, Math.Max (self, min));
          }
  }
}
```

В программном коде приведен метод Clamp, на который производится ссылка из программного кода, приведенного в файле PinchToZoomContainer.cs. Метод применяется для выполнения геометрических преобразований управления, на который будет действовать жест сжатия.

Далее в диалоговом окне «Обозреватель решений» выделить и открыть файл PinchToZoomContainer.cs и далее набрать приведенный ниже программный код Си Шарп

```
using System;
using Xamarin.Forms;

namespace PinchDemo
{
  public class PinchToZoomContainer: ContentView
```

```
  {
        double currentScale = 1;
        double startScale = 1;
        double xOffset = 0;
        double yOffset = 0;
        public PinchToZoomContainer ()
            {
               var pinchGesture = new PinchGestureRecognizer ();
               pinchGesture.PinchUpdated += OnPinchUpdated;
               GestureRecognizers.Add (pinchGesture);
             }
        void OnPinchUpdated
                  (object sender, PinchGestureUpdatedEventArgs e)
          {
                if (e.Status == GestureStatus.Started)
                   {
// Сохранение текущего значения
// масштабного коэффициента,
// примененного к элементу управления
// пользовательского интерфейса
// Обнуление значений центральной
// точки преобразования сжатия.
                    startScale = Content.Scale;
                   Content.AnchorX = 0;
                   Content.AnchorY = 0;
                       }
                  if (e.Status == GestureStatus.Running)
                     {
// Вычисление масштабного коэффициента,
// который будет применен.
                   currentScale += (e.Scale - 1) * startScale;
                   currentScale = Math.Max (1, currentScale);
                   double renderedX = Content.X + xOffset;
                   double deltaX = renderedX / Width;
                   double deltaWidth =
                            Width / (Content.Width * startScale);
                   double originX =
                           (e.ScaleOrigin.X - deltaX) * deltaWidth;
                    double renderedY = Content.Y + yOffset;
                    double deltaY = renderedY / Height;
                    double deltaHeight =
                              Height / (Content.Height * startScale);
                    double originY =
                            (e.ScaleOrigin.Y - deltaY) * deltaHeight;
// Вычислить координаты преобразованного
```

```
// элемента управления в пикселях.
                double targetX =
                        xOffset - (originX * Content.Width) *
                                (currentScale - startScale);
                 double targetY =
                        yOffset - (originY * Content.Height) *
                                (currentScale - startScale);
// Применить преобразование на основе
// изменения исходного состояния.
                  Content.TranslationX =
                        targetX.Clamp(-Content.Width *
                                        (currentScale - 1), 0);
                   Content.TranslationY =
                        targetY.Clamp(-Content.Height *
                                       (currentScale - 1), 0);
// Применить масштабный коэффициент
                    Content.Scale = currentScale;
                  }
                  if (e.Status == GestureStatus.Completed)
                  {
// Сохранение параметров, характеризующих
// изменение элемента управления
                        xOffset = Content.TranslationX;
                        yOffset = Content.TranslationY;
                   }
            }
      }
}
```

В программном коде класс PinchToZoomContainer является дочерним от класса ContentView. Класс служит оболочкой для элемента управления Image на странице MainPage, внутри которого масштабируется изображение (увеличивается или уменьшается) в ответ на жест сжатия.

Масштабирование картинки осуществляется с помощью метода OnPinchUpdated. Этот метод изменяет масштаб элемента управления Image, заключенного в оболочку.

Масштаб, применяемый в месте сжатия, рассчитывается исходя из значений свойств Scale, ScaleOrigin и Status параметра PinchGestureUpdatedEventArgs.

Сначала производится сохранение текущего значения масштабного коэффициента startScale, примененного к элементу управления пользовательского интерфейса, и обнуление значений центральной точки преобразования сжатия. Затем производится вычисление масштабного коэффициента currentScale, который будет применен к элементу управления Image.

С помощью ScaleOrigin производится учет относительных координат элемента управления, на который воздействует жест сжатия. Получаются значения в пикселях для координат X и Y.

Затем производится определение координат targetX и target Y преобразованного элемента управления в пикселях.

С использованием метода Clamp, программный код которого приведен в файле DoubleExtensions.cs, происходит преобразование на основе изменения исходного состояния элемента управления Image, присвоение значений свойствам TranslationX, Translation, а также применение масштабного коэффициента currentScale.

Производится сохранение значений параметров xOffset и yOffset, характеризующих изменение элемента управления.

Далее в диалоговом окне «Обозреватель решений» необходимо выделить и открыть файл MainPage.xaml, в котором необходимо набрать программный код XAML, приведенный далее.

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentPage xmlns=http://xamarin.com/schemas/2014/forms
        xmlns:x=http://schemas.microsoft.com/winfx/2009/xaml
         xmlns:local=
             "clr-namespace:PinchDemo;assembly=PinchDemo"
          x:Class="PinchDemo.MainPage"
          BackgroundColor="YellowGreen">
   <ContentPage.Content>
     <Grid Padding="100">
       <local:PinchToZoomContainer>
         <local:PinchToZoomContainer.Content>
           <Image Source="PinchImage.jpg" />
         </local:PinchToZoomContainer.Content>
       </local:PinchToZoomContainer>
     </Grid>
   </ContentPage.Content>
</ContentPage>
```

В программном коде задается страница типа ContentPage, в которой с помощью свойства BackgroundColor задан цвет фона. Внутри страницы с помощью свойства Content размещается макет Grid. Внутри макета Grid размещается макет ContentView, который создается с помощью программного кода, расположенного в универсальном классе-оболочке PinchToZoomContainer, определенном ранее в программном коде Си Шарп в файле PinchToZoomContainer.cs.

Элемент управления ContentView содержит один дочерний элемент управления Image, в который загружается изображение из файла PinchImage.jpg, находящегося в папке Drawable в проекте PinchDemo.Android.

Для демонстрации возможностей мобильного приложения по работе с жестом сжатия сначала необходимо собрать решение, а затем, при отсутствии ошибок, запустить мобильное программное приложение. Так как с помощью

эмулятора и мыши невозможно реализовать выполнение жеста сжатия, данное программное приложение, в отличие от предыдущих программных приложений, должно быть запущено непосредственно на мобильном устройстве. Порядок действий для настойки запуска мобильного программного приложения непосредственно на мобильном устройстве рассмотрен ранее в разделе 1 учебного пособия.

После запуска мобильного программного приложения в пользовательском интерфейсе отображается картинка, размеры которой можно увеличивать или уменьшать при помощи жеста, выполняемого двумя пальцами.

Далее производится изучение возможности мобильных программных приложений по работе с жестом сдвига с помощью класса PanGestureRecognizer. Жест сдвига используется для обнаружения движения пальцев по экрану при смещении по горизонтали или вертикали содержимого, размещенного в элементе управления, в том случае, если оно не помещается полностью в элементе управления.

Чтобы элемент управления пользовательского интерфейса был доступен для жеста сдвига, необходимо выполнить два пункта:

1. Создать экземпляр класса PanGestureRecognizer.

2. Реализовать обработку события PanUpdated и добавить новый распознаватель жестов в коллекцию GestureRecognizers элемента управления пользовательского интерфейса.

Далее приведен фрагмент программного кода Си Шарп, в котором рассматривается пример использования распознавателя жеста сдвига PanGestureRecognizer, присоединенного к элементу управления Image.

```
var panGesture = new PanGestureRecognizer();
panGesture.PanUpdated += (s, e) =>
{
// Программный код для обработки жеста сдвига
};
image.GestureRecognizers.Add(panGesture);
```

Работа с распознавателем жеста сдвига может быть реализована и с помощью программного кода XAML. Далее приведен фрагмент программного кода, в котором рассмотрен пример использования распознавателя жеста PanGestureRecognizer. Программный код производит обращение к

обработчику OnPanUpdated события PanUpdated, возникающего в случае сдвига картинки, загруженной из файла PanImage.jpg и расположенной внутри элемента управления Image. Программный код обработчика OnPanUpdated содержится в файле с расширением xaml.cs.

```
<Image Source="PanImage.jpg">
 <Image.GestureRecognizers>
  <PanGestureRecognizer PanUpdated="OnPanUpdated" />
 </Image.GestureRecognizers>
</Image>
```

Для выполнения преобразований в ответ на выполнение жеста сдвига в пользовательском интерфейсе требуется выполнения ряда операций, которые используются для сдвига содержимого элемента управления в пределах его границ.

Для этого используется универсальный вспомогательный класс, который заключает в оболочку элемент управления, на содержимое которого будет воздействовать жест сдвига. В презентации приводится фрагмент программного кода Си Шарп, в котором рассматривается использование универсального класса с именем PanContainer, который является дочерним от класса ContentView и ссылается на обработчика OnPanUpdated события PanUpdated, которое происходит в случае выполнения сдвига содержимого внутри элемента управления пользовательского интерфейса.

```
public class PanContainer: ContentView
{ public PanContainer ()
 { var panGesture = new PanGestureRecognizer ();
  panGesture.PanUpdated += OnPanUpdated;
  GestureRecognizers.Add (panGesture);
 }
 void OnPanUpdated (object sender, PanUpdatedEventArgs e)
 {
  // Программный код для обработки жеста сдвига
 }
}
```

Такой универсальный вспомогательный класс может служить оболочкой для элемента управления, содержимое которого масштабируется в ответ на жест сдвига.

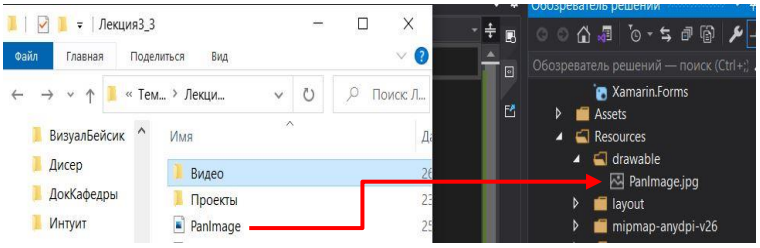
Далее приведен фрагмент программного кода XAML, в котором показан пример использования приведенного ранее класса PanContainer в качестве оболочки для элемента управления Image, на содержимое которого воздействует жест сдвига. В качестве содержимого используется картинка, загруженная из файла PanImage.jpg.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
         xmlns:local="clr-namespace:PanGesture"
         x:Class="PanGesture.HomePage">
  <ContentPage.Content>
```

```
        <AbsoluteLayout>
            <local:PanContainer>
                <Image Source="PanImage.jpg" />
            </local:PanContainer>
        </AbsoluteLayout>
    </ContentPage.Content>
</ContentPage>
```

Для демонстрации возможностей мобильных программных приложений по работе с жестом сдвига с использованием класса PanGestureRecognizer создается проект PanDemo для мультиплатформенного приложения Xamarin.Forms с использованием шаблона «Пустой». Сначала необходимо зайти в диалоговое окно «Обозреватель решений», выделить и открыть проект PanDemo и с помощью рассмотренных ранее действий создать в проекте PanDemo файл класса PanContainer.

Далее необходимо в диалоговом окне «Обозреватель решений», выделить и открыть проект PanDemo.Android, открыть папку Resources и далее перенести файл файл PanImage.jpg из внешнего источника в папку Drawable в проекте PanDemo.Android.



Далее необходимо в диалоговом окне «Обозреватель решений» выделить и открыть проект PanDemo, открыть файл App.xaml.cs и скорректировать имеющийся в нем программный код Си Шарп класса App в соответствии с приведенным ниже программным кодом.

```
using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace PanDemo
{
    public partial class App: Application
    {
        public static double ScreenWidth;
        public static double ScreenHeight;
        public App()
        {
            InitializeComponent();
            MainPage = new MainPage();
        }
    }
```

```
}
```

В программном коде объявлены свойства ScreenWidth и ScreenHeight, которые предназначены для последующей работы с высотой и шириной экрана устройства в программном коде, приведенном в файле PanContainer.cs.

Далее необходимо в диалоговом окне «Обозреватель решений» выделить и открыть проект PanDemo, открыть файл PanContainer.cs и также набрать приведенный далее программный код.

```
using System;
using Xamarin.Forms;
namespace PanDemo
{
        public class PanContainer: ContentView
        {
              double x, y;
              public PanContainer ()
              {
// Настройка PanGestureRecognizer.TouchPoints,
// для того, чтобы контролировать количество точек
// касания, необходимых для выполнения жеста Pan
               var panGesture = new PanGestureRecognizer ();
               panGesture.PanUpdated += OnPanUpdated;
              GestureRecognizers.Add (panGesture);
               }
               void OnPanUpdated
                     (object sender, PanUpdatedEventArgs e)
                  {
                       switch (e.StatusType)
                            {
                             case GestureStatus.Running:
// Выполнение сдвига и проверка того, что не произошло
// выхода за пределы элемента управления
                                  Content.TranslationX =
                          Math.Max (Math.Min (0, x + e.TotalX),
                                  -Math.Abs (Content.Width –
                                        App.ScreenWidth));
                                Content.TranslationY =
                          Math.Max (Math.Min (0, y + e.TotalY),
                                    -Math.Abs (Content.Height –
                                          App.ScreenHeight));
                          break;
                                case GestureStatus.Completed:
// Сохранение преобразования,
//полученного по результатам жеста сдвига
                                  x = Content.TranslationX;
                                  y = Content.TranslationY;
```

```
                    break;
                    }
                }
            }
        }
```

Класс PanContainer выполняет роль универсального класса-оболочки для элемента управления пользовательского интерфейса, на который воздействуют с помощью жеста сдвига.

Жест сдвига предполагает выполнение перемещения содержимого, расположенного внутри элемента управления пользовательского интерфейса. При этом содержимое не помещается полностью в элементе управления. Поэтому для того, чтобы просмотреть полностью содержимое, пользователь перемещает его внутри элемента управления.

В программном коде сначала производится создание экземпляра класса PanGestureRecognizer как дочернего класса от класса ContentView. Затем производится добавление нового распознавателя жестов в коллекцию GestureRecognizers элемента управления Image с помощью GestureRecognizers.

Для обработки события сдвига используется обработчик OnPanUpdated, который обновляет видимое содержимое элемента управления в соответствии с жестом сдвига пользователя.

Определение положения содержимого, которое отображается, внутри элемента управления проверка того, что не произошло выхода содержимого за пределы элемента управления, производится в процессе определения значений свойств TranslationX и TranslationY.

Значение этих свойств рассчитываются с использованием значений свойств TotalX и TotalY параметра PanUpdatedEventArgs для обработчика OnPanUpdated. Также для расчета значений свойств TranslationX и TranslationY используются значения свойств App.ScreenWidth и App.ScreenHeight, которые соответствуют высоте и ширине окна просмотра. В качестве их значений задаются высота и ширина экрана устройства в зависимости от мобильного устройства, на котором запущено мобильное программное приложение.

После окончания сдвига содержимого элемента управления производится сохранение преобразований, полученных по результатам жеста сдвига. Значениям x и y присваиваются значения свойств TranslationX и TranslationY.

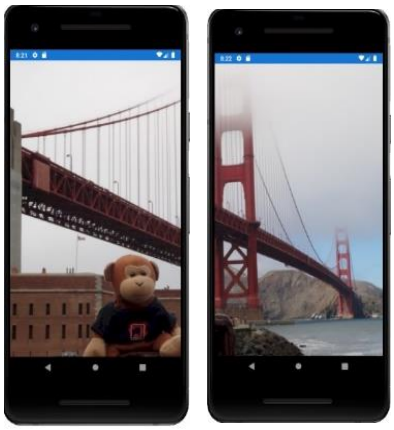
Далее необходимо в диалоговом окне «Обозреватель решений» выделить и открыть проект PanDemo, выделить и открыть файл MainPage.xaml и набрать приведенный ниже программный код XAML, который задает страницу MainPage, которую увидит пользователь при запуске программного приложения.

```
<ContentPage
    xmlns="http://xamarin.com/schemas/2014/forms"
    xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
    xmlns:local="clr-namespace:PanDemo"
    x:Class="PanDemo.MainPage">
```

```
<ContentPage.Content>
    <AbsoluteLayout>
        <local:PanContainer>
            <Image Source="PanImage.jpg"
                   WidthRequest="1024"
                   HeightRequest="768" />
        </local:PanContainer>
    </AbsoluteLayout>
</ContentPage.Content>
</ContentPage>
```

В программном коде страница типа ContentPage содержит макет AbsolutLayout, внутри которого располагается элемент управления Image. Элемент управления Image заключен в оболочку с помощью класса PanContainer, который с помощью реализуемого им макета ContentView делает возможным сдвиг картинки из файла PanImage.jpg внутри элемента управления Image. Свойствам WidthRequest и HeightRequest присваиваются значения, характеризующие ширину и высоту элемента управления Image. При этом размеры изображения в файле PanImage.jpg превышают размеры элемента управления Image. Поэтому изображение внутри элемента управления Image будет передвигаться с помощью жеста сдвига для просмотра всего изображения.

Для демонстрации возможностей мобильных приложений по работе с жестом сдвига сначала необходимо собрать решение, а затем, при отсутствии ошибок, запустить программное приложение для работы с жестом сдвига. После запуска мобильного программного приложения отображается страница, в которой размещен элемент управления Image. Картинка, являющаяся содержимым элемента управления Image, не помещается полностью внутри элемента управления, и поэтому часть картинки скрыта от пользователя.



С помощью макета ContentView, который реализуется с помощью класса PanContainer, имеется возможность сдвига картинки внутри элемента управления Image для выполнения ее полного просмотра. Сдвиг картинки в пользовательском интерфейсе может осуществляться в различных направлениях.

Далее рассматриваются возможности мобильных программных приложений по работе с жестом прокрутки с помощью класса SwipeGestureRecognizer. Жест прокрутки выполняется для просмотра содержимого элемента пользовательского интерфейса (если оно не помещается полностью в пределах элемента управления) и реализуется с помощью перемещения пальца по сенсорному экрану в горизонтальном или вертикальном направлении.

Чтобы реализовать доступность элемента управления пользовательского интерфейса для жеста прокрутки, необходимо выполнить несколько пунктов:

1. Создать экземпляр класса SwipeGestureRecognizer.

2. Задать значение свойства Direction, а также, при необходимости, значение свойства Threshold.

3. Реализовать обработку события Swiped и добавить новый распознаватель жестов в коллекцию GestureRecognizers элемента управления пользовательского интерфейса.

Далее приведен фрагмент программного кода Си Шарп, в котором рассматривается пример использования распознавателя жеста прокрутки SwipeGestureRecognizer, присоединенного элементу управления BoxView красного цвета.

Направление прокрутки, заданное с помощью свойства Direction, соответствует прокрутке вправо по горизонтали. Показано, что при возникновении события Swiped срабатывает обработчик OnSwiped.

```
var boxView = new BoxView {Color = Color.Red};
var rightSwipeGesture =
                        new SwipeGestureRecognizer
                       {Direction = SwipeDirection.Right};
rightSwipeGesture.Swiped += OnSwiped;
boxView.GestureRecognizers.Add(rightSwipeGesture);
```

Работа с распознавателем жеста прокрутки может быть реализована и с помощью программного кода XAML. Далее приведен фрагмент программного кода XAML, в котором рассматривается пример использования распознавателя жеста прокрутки SwipeGestureRecognizer, присоединенного к элементу управления BoxView. При этом направления прокрутки, заданные с помощью свойства Direction, соответствует прокрутке влево и вправо по горизонтали.

Программный код производит обращение к обработчику OnSwiped события Swiped, возникающего в случае прокрутки элемента управления BoxView зеленого цвета.

Программный код обработчика OnSwiped должен содержаться в файле с расширением xaml.cs. При этом файл с расширением xaml.cs сопряжен в проекте с файлом с расширением xaml, из которого производится обращение к обработчику OnSwiped.

```
<BoxView Color="Green">
  <BoxView.GestureRecognizers>
    <SwipeGestureRecognizer
```

```
            Direction="Left, Right"
            Swiped="OnSwiped"/>
    </BoxView.GestureRecognizers>
</BoxView>
```

Значение свойства Direction выбирается с использованием перечисления SwipeDirection. Для свойства Direction могут быть заданы одновременно несколько значений из перечисления SwipeDirection для того, чтобы событие Swiped происходило в случае жеста прокрутки более чем в одном направлении. При этом действует ограничение, заключающееся в том, что отдельный распознаватель SwipeGestureRecognizer может распознавать жесты прокрутки только по одной оси (горизонтальной или вертикальной).

При необходимости может быть задано значение свойства Threshold. Свойство Threshold представляет собой минимальное, в аппаратно-независимых единицах, расстояние прокрутки, необходимое для распознавания жеста прокрутки. Значение свойства по умолчанию для этого свойства равно 100. Это означает, что любая прокрутка менее 100 аппаратно-независимых единиц будет игнорироваться.

Для демонстрации возможностей мобильных приложений по работе с жестом прокрутки создается проект SwipeDemo для мультиплатформенного приложения Xamarin.Forms с использованием шаблона «Пустой».

Сначала необходимо зайти в диалоговое окно «Обозреватель решений», выделить и открыть проект SwipeDemo, выделить и открыть файл MainPage.xaml и затем набрать приведенный далее программный код XAML, который определяет внешний вид пользовательского интерфейса, отображаемого при запуске мобильного программного приложения.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
         x:Class="SwipeDemo.MainPage"
         BackgroundColor="Cyan">
  <StackLayout Margin="20">
    <Label
        Text="Проведите пальцем внутри цветного квадрата"
         TextColor="Black"
         FontSize="30"
         FontAttributes="Bold"
         HorizontalTextAlignment="Center"
         VerticalTextAlignment="Center"/>
    <BoxView Color="Orange"
             WidthRequest="400"
             HeightRequest="400"
             HorizontalOptions="Center"
             VerticalOptions="CenterAndExpand">
      <BoxView.GestureRecognizers>
        <SwipeGestureRecognizer Direction="Left"
                                Swiped="OnSwiped"/>
```

```
            <SwipeGestureRecognizer Direction="Right"
                                    Swiped="OnSwiped"/>
            <SwipeGestureRecognizer Direction="Up"
                                    Swiped="OnSwiped"/>
            <SwipeGestureRecognizer Direction="Down"
                                    Swiped="OnSwiped"/>
          </BoxView.GestureRecognizers>
        </BoxView>
        <Label x:Name="lab"
            Text="Направление прокрутки: "
            TextColor="Black"
            FontSize="30"
            FontAttributes="Bold"
            HorizontalTextAlignment="Center"
            VerticalTextAlignment="Center"/>
    </StackLayout>
</ContentPage>
```

Программный код задает страницу типа ContentPage, в которой с помощью свойства BackgroundColor задан цвет фона. Внутри страницы расположен макет StackLout.

В макете StackLout содержатся два элемента управления Label и элемент управления BoxView.

С помощью элемента управления BoxView отображается квадрат с заданной шириной, высотой и цветом. К элементу управления BoxView будет применяться жест прокрутки. Поэтому к нему привязаны четыре распознавателя прокрутки для распознавания прокрутки влево, вправо по горизонтали, а также вверх и вниз по вертикали.

В случае возникновения события Swiped производится обращение к обработчику OnSwiped, программный код которого приведен в файле MainPage.xaml.cs.

В первом элементе управления Label отображается вспомогательный текст о необходимости провести пальцем по элементу управления BoxView.

Во втором элементе управления Label с именем lab после выполнения жеста прокрутки выдается информация о выполнении жеста прокрутки в пределах элемента управления BoxView, а также о направлении прокрутки. С помощью атрибута x:Name имя элемента управления передается в программный код обработчика OnSwiped.

В диалоговом окне «Обозреватель решений» в проект SwipeDemo, необходимо выделить и открыть файл MainPage.xaml.cs, связанный с файлом MainPage.xaml и затем набрать приведенный далее программный код Си Шарп.

```
using Xamarin.Forms;
namespace SwipeDemo
{   public partial class MainPage: ContentPage
    {
```

```
        public MainPage()
        {
            InitializeComponent();
        }
        void OnSwiped(object sender, SwipedEventArgs e)
        {   lab.Text =
            $" Направление прокрутки: {e.Direction.ToString()}";
        }
    }
}
```

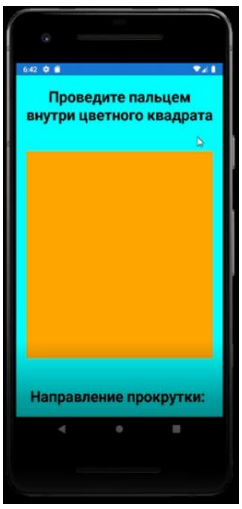
Программный код класса MainPage содержит обработчик события OnSwiped, который через переменную e типа SwipedEventArgs принимает аргументы жеста прокрутки (в частности, Direction - направление прокрутки) и выдает в элемент управления Label с именем lab информацию о направлении, в котором выполнен жест прокрутки.

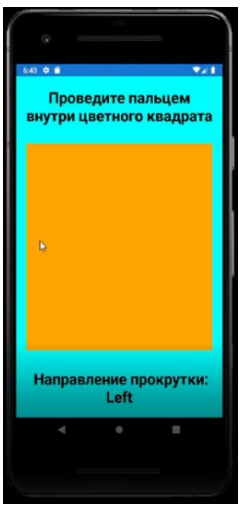
Для демонстрации возможностей мобильных программных приложений по работе с жестом прокрутки необходимо сначала собрать решение, а затем, при отсутствии ошибок, запустить программное приложение.

После запуска мобильного программного приложения в пользовательском интерфейсе запущенного приложения отображается текст, который напоминает о необходимости провести пальцем внутри цветного квадрата. В момент запуска программного приложения под цветным квадратом, не отображается информация о направлении прокрутки. Обработка жеста прокрутки в пользовательском интерфейсе производится только при проведении пальцем внутри цветного квадрата.

Проведение пальцем вне цветного квадрата не приведет к обработке жеста прокрутки.

При обработке жеста прокрутки под цветным квадратом выдается информация о направлении прокрутки. Прокрутка может проводиться влево и вправо по горизонтали, а также вверх и вниз по вертикали.





Далее производится рассмотрение возможностей мобильных программных приложений по работе с жестом перетаскивания. Жест перетаскивания позволяет перемещать элемент пользовательского интерфейса и связанный с ним пакет данных из одной точки на пользовательском интерфейсе в другую. Источником перетаскивания является элемент пользовательского интерфейса, на котором инициируется жест перетаскивания (то есть, который необходимо перетащить из одного места пользовательского интерфейса в другое, целевое, место). Источник перетаскивания производит передачу связанных с ним данных в пакет данных.

Если перетаскивание источника в целевое место пользовательского интерфейса (в цель перетаскивания) завершено, то происходит отпускание источника перетаскивания в целевую точку отпускания на пользовательском интерфейсе программного приложения.

Целевая точка отпускания обрабатывает пакет данных и в результате получает элемент пользовательского интерфейса, являвшийся ранее источником перетаскивания.

Работа мобильного программного приложения с жестом перетаскивания состоит из следующих действий:

1. Включение доступа жеста перетаскивания к элементу управления-источнику перетаскивания за счет добавления для элемента управления распознавателя DragGestureRecognizer в коллекцию GestureRecognizers, а также выбор для свойства DragGestureRecognizer.CanDrag значения true.

2. Создание пакета данных. При этом Xamarin.Forms автоматически формирует пакет данных для ряда элементов управления пользовательского интерфейса, и поэтому создания для них пакета данных с помощью программного кода не требуется. Но для других элементов управления пользовательского интерфейса требуется дополнительное формирование программного кода для создания пакета данных.

3. Включение доступности для отпускания элемента управления-источника перетаскивания за счет добавления распознавателя DropGestureRecognizer в коллекцию GestureRecognizers, а также выбор для свойства DropGestureRecognizer.AllowDrop значения true. Включение производится при «зависании» источника перетаскивания над целью перетаскивания (то есть, над тем местом пользовательского интерфейса, в которое должен быть размещен перетаскиваемый элемент управления).

4. Обработка события DragOver, соответствующего окончанию перетаскивания и «зависанию» над целью перетаскивания.

5. Обработка созданного ранее пакета данных для получения данных от элемента управления-источника. При этом Xamarin.Forms будет автоматически обрабатывать пакеты данных для для ряда элементов управления пользовательского интерфейса, и поэтому разработка программного кода для обработки пакета данных не требуется. Для других элементов управления пользовательского интерфейса требуется разработка программного кода для обработки пакета данных.

Распознавание жеста перетаскивания обеспечивается классом DragGestureRecognizer, который имеет следующие свойства, методы и события:

свойство CanDrag определяет, может ли элемент управления, к которому прикреплен распознаватель жестов, быть источником перетаскивания (по умолчанию значение равно false);

свойство DragStartingCommand задает команду, которая выполняется при первом распознавании жеста перетаскивания;

свойство DragStartingCommandParameter задает параметр, который передается в команду, указанную в свойстве DragStartingCommand;

свойство DropCompletedCommmand задает команду, которая выполняется при отпускании источника перетаскивания;

свойство DropCompletedCommandParameter задает параметр, который передается в команду, указанную в свойстве DropCompletedCommand.

Класс DragGestureRecognizer также определяет события DragStarting и DropCompleted.

Когда распознаватель DragGestureRecognizer обнаруживает жест перетаскивания, он выполняет команду, указанную в свойстве DragStartingCommand, и вызывает событие DragStarting. Далее, когда распознаватель DragGestureRecognizer обнаруживает отпускание источника перетаскивания, он выполняет команду, указанную в свойстве DropCompletedCommand, и вызывает событие DropCompleted.

Объект DragStartingEventArgs, соответствующий событию DragStarting, имеет следующие свойства:

свойство Handled указывает, обрабатывал ли обработчик событие или следует ли продолжать обработку;

свойство Cancel указывает на необходимость отмены события;

свойство Data доступно только для чтения и содержит пакет данных, сопровождающий источник перетаскивания.

Объект DropCompletedEventArgs, соответствующий событию DropCompleted, имеет свойство DropResult, доступное только для чтения.

Далее приведен фрагмент программного кода XAML, в котором показан распознаватель DragGestureRecognizer, присоединенный к элементу управления Image, в который загружено изображение из файла DragImage.jpg. При этом элемент управления Image доступен для перетаскивания.

```
<Image Source="DragImage.png">
   <Image.GestureRecognizers>
      <DragGestureRecognizer CanDrag="True" />
   </Image.GestureRecognizers>
</Image>
```

Создание пакета данных происходит автоматически при выполнении перетаскивания для следующих элементов управления:

1. Текстовые значения могут перетаскиваться из элементов управления CheckBox, DatePicker, Editor, Entry, Label, RadioButton, Switch и TimePicker.

2. Изображения можно перетаскивать из элементов управления Button, Image и ImageButton.

Для остальных элементов управления необходима разработка программного кода для создания пакета данных.

Пакеты данных формируются с использованием класса DataPackage, который имеет следующие свойства:

свойство Properties представляет собой коллекцию данных, содержащихся в DataPackage (свойство доступно только для чтения);

свойство Image задает изображение, содержащееся в DataPackage;

свойство Text задает текст, содержащийся в DataPackage;

свойство View задает версию DataPackage и предназначено только для чтения.

Изображения или текстовые данные могут быть связаны с источником перетаскивания путем сохранения данных в свойстве DataPackage.Image или DataPackage.Text. Это делается в обработчике события DragStarting.

В приведенном далее фрагменте программного кода XAML распознаватель DragGestureRecognizer присоединяется к объекту Path. Событие DragStarting срабатывает при обнаружении жеста перетаскивания объекта Path, что приводит к выполнению обработчика OnDragStarting.

```
<Path Stroke="Black">
  <Path.GestureRecognizers>
    <DragGestureRecognizer CanDrag="True"
                DragStarting="OnDragStarting" />
  </Path.GestureRecognizers>
  <Path.Data>
    <!—Данные объекта Path -->
  </Path.Data>
</Path>
```

Событию DragStarting соответствует объект е типа DragStartingEventArgs, который в качестве свойства принимает объект Data типа DataPackage. Программный код Си Шарп для обработчика OnDragStarting приведен далее. Свойству Text объекта Data типа DataPackage присваивается текстовая строка. При работе с источником перетаскивания, то есть с объектом Path, можно обращаться к объекту Data для получения доступа к текстовой строке.

```
void OnDragStarting(object sender, DragStartingEventArgs e)
{e.Data.Text = "Текстовые данные объекта Path";}
```

Такие данные, как изображения и текст, характеризующие источник перетаскивания, могут быть сохранены с помощью обработчика события DragStarting в коллекции DataPackage.Properties.

Далее приведен фрагмент программного кода XAML, в котором рассматривается распознаватель DragGestureRecognizer, регистрирующий обработчик для события DragStarting. Распознаватель DragGestureRecognizer присоединен к объекту Rectangle. Событие DragStarting срабатывает при обнаружении жеста перетаскивания объекта Rectangle, что приводит к выполнению обработчика событий OnDragStarting.

```
<Rectangle Stroke="Red" Fill="Dark Blue"
        HeightRequest="200" WidthRequest="200">
   <Rectangle.GestureRecognizers>
      <DragGestureRecognizer CanDrag="True"
                     DragStarting="OnDragStarting" />
   </Rectangle.GestureRecognizers>
</Rectangle>
```

В приведенном далее фрагменте программного кода Си Шарп приведен обработчик события OnDragStarting, на который ссылается приведенный ранее программный код XAML. Объект e типа DragStartingEventArgs, соответствующий событию DragStarting, имеет свойство Data с типом DataPackage. Коллекция Properties применяется для хранения необходимых данных объекта Square, который представляет размер Rectangle.

```
void OnDragStarting(object sender, DragStartingEventArgs e)
{
   Shape shape = (sender as Element).Parent as Shape;
   e.Data.Properties.Add
        ("Square", new Square(shape.Width, shape.Height));
}
```

Распознавание жеста отпускания производится с помощью класса DropGestureRecognizer, который имеет следующие свойства:

свойство AllowDrop определяет, может ли быть отпущен источник перетаскивания (элемент управления), к которому прикреплен распознаватель жестов, на целевую точку отпускания на пользовательском интерфейсе (по умолчанию значение равно false);

свойство DragOverCommand задает команду, которая выполняется при завершении перетаскивания источника перетаскивания (при его «зависании» над целевой точкой на пользовательском интерфейсе);

свойство DragOverCommandParameter задает параметр, передаваемый в команду, указанную в свойстве DragOverCommand;

свойство DropCommand задает команду, выполняемую при отпускании источника перетаскивания над целевой точкой на пользовательском интерфейсе;

свойство DropCommandParameter задает параметр, который передается в команду, указанную в свойстве DropCommand.

Класс DropGestureRecognizer также определяет события DragOver и Drop. Когда распознаватель DropGestureRecognizer распознает нахождение источника перетаскивания над целевой точкой отпускания на пользовательском интерфейсе, он выполняет команду, указанную в свойстве DragOverCommand, и вызывает событие DragOver.

Событию DragOver соответствует класс DragEventArgs, который имеет следующие свойства:

свойство Data доступно только для чтения и содержит данные, связанные ранее с источником перетаскивания;

свойство AcceptedOperation указывает, какие операции разрешены целевым элементом отпускания в случае отпускания источника перетаскивания.

Свойство AcceptedOperation может принимать следующие значения:

значение None указывает, что действие не будет выполнено;

значение Copy указывает, что содержимое источника перетаскивания будет скопировано в целевую точку отпускания.

Далее приведен фрагмент программного кода XAML, в котором показан распознаватель DropGestureRecognizer, присоединенный к элементу управления Image, который обращается к обработчику OnDragOver для события DragOver.

```
<Image BackgroundColor="Silver"
    HeightRequest="300" WidthRequest="250">
  <Image.GestureRecognizers>
    <DropGestureRecognizer AllowDrop="True"
                DragOver="OnDragOver" />
  </Image.GestureRecognizers>
</Image>
```

Фрагмент программного кода Си Шарп для обработчика OnDragOver приведен далее. Событие DragOver происходит, когда источник перетаскивания находится над целевой точкой отпускания на пользовательском интерфейсе, но при этом пока еще не отпущен на целевую точку. Свойству AcceptedOperation объекта e типа DragEventArgs присваивается значение None.

Таким образом, при перемещении источника перетаскивания целевой точкой на пользовательском интерфейсе далее не выполняется никаких действий.

```
void OnDragOver(object sender, DragEventArgs e)
{
  e.AcceptedOperation = DataPackageOperation.None;
}
```

Если DropGestureRecognizer распознает отпускание источника перетаскивания над целевой точкой на пользовательском интерфейсе, он производит выполнение команды, указанной в свойстве DropCommand, и вызывает событие Drop.

Событию Drop соответствует объект e типа DropEventArgs, который имеет следующие свойства:

свойство Data доступно только для чтения и содержит данные, связанные ранее с источником перетаскивания;

свойство Handled указывает, следует ли продолжать дальнейшую обработку события.

При срабатывании события Drop источник перетаскивания копируется в целевую точку отпускания на пользовательском интерфейсе. При этом производится автоматическое получение данных из пакета данных из

следующих элементов управления, которые могут являться источниками перетаскивания:

1. Текстовые данные можно получать из элементов управления CheckBox, DatePicker, Editor, Entry, Label, RadioButton, Switch и TimePicker.

2. Изображения можно получать из элементов управления Button, Image и ImageButton.

Для других типов данных и элементов управления необходимо самостоятельно разрабатывать программный код для работы с пакетом данных.

Объект e типа DropEventArgs, который соответствует событию Drop, имеет свойство в виде объекта Data типа DataPackageView, доступное только для чтения и содержащее данные, связанные с источником перетаскивания.

Изображения или текстовые данные можно получить из пакета данных в обработчике события Drop с помощью методов GetImageAsync и GetTextAsync, определенных в классе DataPackageView. Метод GetImageAsync извлекает изображение из пакета данных, хранимого в свойстве DataPackage.Image. Метод GetTextAsync также извлекает текст из пакета данных, хранимого в свойстве DataPackage.Text.

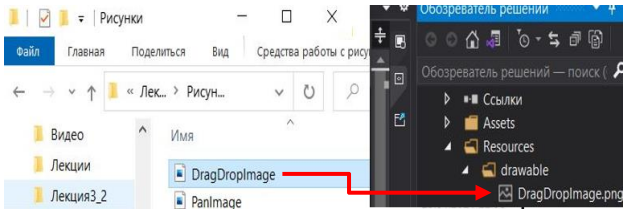
Далее приведен фрагмент программного кода Си Шарп для обработчика события Drop, который извлекает текст из пакета данных с помощью метода GetTextAsync.

```
async void OnDrop(object sender, DropEventArgs e)
{
   string text = await e.Data.GetTextAsync();
   // Программный код для обработки данных
}
```

Также данные из пакета данных можно получить, обратившись к коллекции свойств Properties пакета данных. Далее приведен фрагмент программного кода Си Шарп для обработчика события Drop, который извлекает данные из пакета данных для объекта Square из коллекции свойств Properties пакета данных с помощью указания ключа «Square».

```
void OnDrop(object sender, DropEventArgs e)
{   Square square = (Square)e.Data.Properties["Square"];
   // Программный код для обработки данных
}
```

Для демонстрации возможностей мобильного программного приложения по работе с распознавателем жестов перетаскивания и отпускания создается проект DragDropDemo для мультиплатформенного приложения Xamarin.Forms с использованием шаблона «Пустой». Сначала неоьходимо зайти в диалоговое окно «Обозреватель решений», выделить и открыть проект DragDropDemo.Android и далее открыть пвпку Resources и перенести файл DragDropImage.png из внешнего источника в папку Drawable в проекте DragDropDemo.Android.



Далее в диалоговом окне «Обозреватель решений» необходимо выделить и открыть проект PinchDemo, открыть файл App.xaml.cs и скорректировать программный код в этом файле. В приведенном ниже программном коде Си Шарп подключается флаг DragAndDrop_Experimental, который делает возможным работу с жестом перетаскивания и отпускания в мобильном программном приложении Xamarin.Forms.

```
using System;
using Xamarin.Forms;
using Xamarin.Forms.Xaml;
namespace DragDropDemo
{   public partial class App : Application
    {   public App()
        {  Device.SetFlags(new string[]
                           { "DragAndDrop_Experimental" });
           InitializeComponent();
           MainPage = new MainPage();
        }
    }
}
```

Далее в диалоговом окне «Обозреватель решений» необходимо выделить и открыть файл MainPage.xaml, после чего набрать приведенный ниже программный код XAML.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
           x:Class="DragDropDemo.MainPage">
    <StackLayout Margin="20">
      <Label x:Name="lab"
           Text="Перенести изображение в корзину с
                          правильным названием"
           FontSize="20"
           FontAttributes="Bold"
           TextColor="Red"
           HorizontalTextAlignment="Center"/>
      <Image x:Name="Pic"
           Source="DragDropImage.png"
           HorizontalOptions="Center">
        <Image.GestureRecognizers>
```

```
            <DragGestureRecognizer CanDrag="True" />
        </Image.GestureRecognizers>
    </Image>
    <Grid ColumnDefinitions="0.5*, 0.5*"
          RowDefinitions="*, *"
          HeightRequest="300"
          VerticalOptions="EndAndExpand">
        <Image BackgroundColor="Yellow"
               HeightRequest="300"
               WidthRequest="300">
            <Image.GestureRecognizers>
                <DropGestureRecognizer
                    AllowDrop="True"
                    Drop="OnCorrectDrop" />
            </Image.GestureRecognizers>
        </Image>
        <Label Grid.Row="1"
          HorizontalOptions="Center"
          Text="Обезьяна"
          FontSize="20"
          FontAttributes="Bold"
          TextColor="Black"
          HorizontalTextAlignment="Center"/>
        <Image x:Name="cat"
          Grid.Column="1"
          BackgroundColor="Yellow"
          HeightRequest="300"
          WidthRequest="300">
            <Image.GestureRecognizers>
                <DropGestureRecognizer
                    AllowDrop="True"
                    DragOver="OnIncorrectDrop" />
            </Image.GestureRecognizers>
        </Image>
        <Label Grid.Row="1"
          Grid.Column="1"
          HorizontalOptions="Center"
          Text="Кошка"
          FontSize="20"
          FontAttributes="Bold"
          TextColor="Black"
          HorizontalTextAlignment="Center"/>
    </Grid>

</StackLayout>
```

</ContentPage>

Программный код задает страницу типа ContentPage, внутри которой располагается макет StackLayout, содержащий следующие элементы управления:

1. Элемент управления Label с именем lab, в котором будет отображается текст о необходимости перетащить картинку в корзину с правильным названием. Также в данном элементе управления будут отображаться сообщения о корректности или некорректности переноса источника перетаскивания в целевые объекты отпускания. Для данного элемента управления заданы значения свойств, характеризующих размер и написание шрифта, цвет букв, а также ориентацию текста по горизонтали внутри элемента управления.

2. Элемент управления Image с именем Pic, в который загружено изображение из файла DragDropImage.png из папки Drawable в проекте DragDropDemo.Android. Данный элемент управления Image будет служить источником перетаскивания. К элементу управления Image привязан распознаватель перетаскивания DragGestureRecognizer. Также с помощью свойства CanDrag сделано возможным перетаскивание элемента управления Image.

3. Макет Grid, внутри которого расположена таблица из двух строк и столбцов. При этом, заданы параметры, характеризующие размеры ячеек таблицы.

В первой строке макета Grid размещаются два элемента управления Image, которые выступают в качестве объектов, в которые отпускается источник перетаскивания, и к которым привязаны распознаватели DropGestureRecognizer. Также с помощью свойства AllowDrop сделано возможным отпускание источника перетаскивания в элементы управления Image, находящихся внутри макета Grid (они соответствуют целевым точкам отпускания на пользовательском интерфейсе). Для обоих элементов управления Image заданы значения свойств, характеризующих их размеры, цвет фона. Одному из элементов управления Image присвоено имя cat (этот элемент управления расположен во втором столбце первой стоки таблицы).

Во второй строке макета Grid находятся два элемента Label, в которых отображаются названия изображений. При этом для данных элементов управления заданы значения свойств, характеризующих размер и написание шрифта, цвет букв, а также ориентацию текста по горизонтали внутри элементов управления.

Если с элементом управления Image в первом столбце макета Grid происходит событие Drop, то производится обращение к обработчику OnCorrectDrop. То есть корректным является перемещение элемента управления Image.

Если с элементом управления Image во втором столбце макета Grid происходит событие Drop, то производится обращение к обработчику OnIncorrectDrop.

Программный код для обработчиков OnCorrectDrop и OnIncorrectDrop приведен в файле MainPage.xaml.cs.

Далее в диалоговом окне «Обозреватель решений» необходимо выделить и открыть файл MainPage.xaml.cs и после этого набрать приведенный далее программный код Си Шарп.

```
using Xamarin.Forms;
namespace DragDropDemo
{
   public partial class MainPage: ContentPage
   {
      public MainPage()
      {
         InitializeComponent();
      }
      void OnCorrectDrop(object sender, DropEventArgs e)
      {
         lab.Text="Корректное перемещение";
         Pic.IsVisible = false;
      }
      void OnIncorrectDrop(object sender, DropEventArgs e)
      {
         lab.Text = "Некорректное перемещение";
         e.Handled = false;
      }
   }
}
```

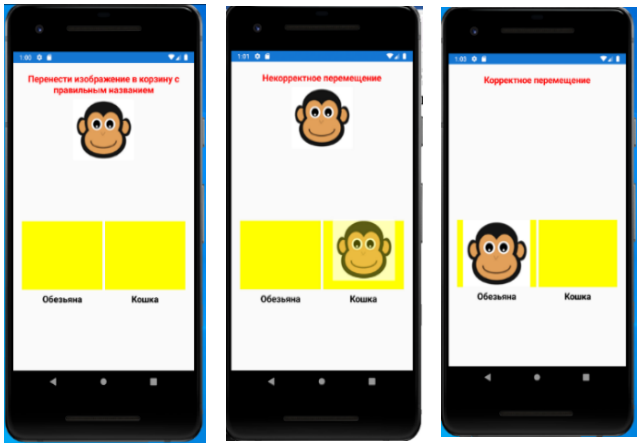
Программный код содержит обработчики событий OnCorrectDrop и OnIncorrectDrop для обработки событий перетаскивания элементов управления Image с загруженными в них изображениями, которые и являются источниками перетаскивания. Элементы управления Image перетаскивается и затем отпускается в целевую точку на пользовательском интерфейсе. Ссылка на обработчиков событий OnCorrectDrop и OnIncorrectDrop производится из файла MainPage.xaml.

Поскольку для свойства AllowDrop элементов управления Image в макете Grid было ранее установлено значение True, то элементы управления Image доступны и для отпускания.

Обработчик OnCorrectDrop события Drop предусматривает отображение в элементе управления с именем lab текста о корректном перенесении источника перетаскивания в целевой объект отпускания, под которым с помощью элемента управления Label отображен текст «Обезьяна». Также обработчик будет выполнять перегрузку источника перетаскивания в целевую точку на пользовательском интерфейсе. При этом, после перегрузки источник перетаскивания (элемент управления Image с именем Pic) становится невидимым, и считается что выполнение задачи перетаскивания завершено.

Обработчик OnIncorrectDrop события Drop указывает с помощью значения свойства Handled, равного false, что событие не будет далее обрабатываться, и перемещение источника перетаскивания в целевую точку отпускания, под которой с помощью элемента управления Label отображен текст «Кошка», не будет выполнено. При этом, производится отображение в элементе управления с именем lab текста о некорректном перенесении источника перетаскивания в целевую точку отпускания на пользовательском интерфейсе. После этого источник перетаскивания не становится невидимым и по-прежнему остается доступным для перетаскивания.

Для демонстрации возможностей мобильного программного приложения по работе с жестами перетаскивания и отпускания сначала необходимо собрать решение, а затем, в случае отсутствия ошибок, запустить программное приложение. После запуска мобильного программного приложения в пользовательском интерфейсе отображается изображение, которое является источником перетаскивания, и две корзины желтого цвета, являющиеся целевыми точками отпускания. В верхней части пользовательского интерфейса отображено напоминание о необходимости перетащить картинку в одну из корзин. Под картинами отображается вспомогательный текст с названиями корзин.



При перетаскивании картинки в корзину с неправильным названием в верхней части пользовательского интерфейса выдается сообщение о некорректном перетаскивании. При этом картинка возвращается на свое место и остается доступной для следующего перетаскивания. При перетаскивании в корзину с правильным названием картинка размещается в корзинке, в верхней части пользовательского интерфейса выдается сообщение о корректном перетаскивании, а источник перетаскивания становится невидимым и недоступным для перетаскивания.