Работа с коллекциями данных с помощью элементов управления ListView и TableView

Рассмотрим возможности мобильных программных приложений по работе со списками и коллекциями с использованием элемента управления ListView, который используется для создания прокручиваемых списков, каждая строка которых является ячейкой, то есть, экземпляром класса ViewCell и предназначена для отображения текста и изображений.

В элементе ViewCell могут отображаться текст и изображения, указываться состояние true или false, а также отображаться введенные пользователем данные.

Существуют встроенные ячейки и пользовательские ячейки. К встроенным ячейкам относятся:

ячейка TextCell, отображающая строку текста (при необходимости - с текстом подробностей в виде второй строки с меньшим шрифтом и контрастным цветом);

ячейка ImageCell, отображающая изображение с поясняющим текстом, при этом ячейка отображается так же, как и ячейка TextCell, но с изображением слева;

ячейка SwitchCell, отображающая состояния «вкл./выкл.» или «true/false»;

ячейка EntryCell, отображающая текстовые данные, вводимые и корректируемые пользователем.

Пользовательские ячейки позволяют создавать макеты ячеек, которые не поддерживаются встроенными ячейками. Пользовательские ячейки, как и встроенные, являются производными от класса ViewCell.

Ячейка TextCell имеет следующие свойства:

свойство Text задает текст, отображаемый в первой строке крупным шрифтом;

свойство Detail задает текст, отображаемый под первой строкой с меньшим шрифтом;

свойство TextColor устанавливает цвет текста, заданный в свойстве Text;

свойство DetailColor устанавливает цвет текста, заданный в свойстве Detail.

Ячейка ImageCell имеет такие же свойства, что и ячейка TextCell, при этом имеется еще одно свойство ImageSource, которое задает файл с изображением, отображаемым рядом с текстом:

свойство Text задает текст, отображаемый в первой строке крупным шрифтом;

свойство Detail задает текст, отображаемый под первой строкой с меньшим шрифтом;

свойство TextColor устанавливает цвет текста, заданный в свойстве Text;

свойство DetailColor устанавливает цвет текста, заданный в свойстве Detail;

свойство ImageSource задает файл с изображением, отображаемым рядом с текстом.

Ячейка SwitchCell имеет следующие свойства:

свойство Text задает текст, отображаемый рядом с переключателем, задающим состояние «вкл./выкл» или «true/false»;

свойство On указывает, отображается ли переключатель в виде «on» или «off»;

свойство OnColor задает цвет переключателя, когда он находится в положении «On».

Ячейке SwitchCell также соответствует событие OnChanged, позволяющее реагировать на изменение состояния переключателя.

Ячейка EntryCell имеет следующие свойства:

свойство Keyboard задает вид клавиатуры (например, числовые значения, электронная почта, номера телефонов и т. д,) отображаемой во время редактирования текста в ячейке;

свойство Label задает текст, отображаемый слева от поля ввода текста;

свойство LabelColor задает цвет текста, который указан в свойстве Label;

свойство Placeholder задает текст, отображаемый в поле ввода ячейки, если в ней не производилось редактирование текста, при этом, этот текст исчезает при начале редактирования;

свойство Text задает текст, находящийся в поле ввода;

свойство HorizontalTextAlignment задает горизонтальное выравнивание текста (текст выравнивается по центру, по левому или по правому краю);

свойство VerticalTextAlignment задает вертикальное выравнивание текста (значения свойства Start, Center, End).

Ячейке EntryCell также соответствует событие Completed, которое возникает, когда пользователь нажимает кнопку «Ввод» на клавиатуре при редактировании текста внутри ячейки.

Далее приведен фрагмент программного кода, в котором представлен пример шаблона пользовательской ячейки, в состав которой входят два элемента управления Label.

```
<ListView.ItemTemplate>
    <DataTemplate>
        <ViewCell>
            <StackLayout
                BackgroundColor="Black"
                Orientation="Horizontal">
                <Label Text="{Binding title}"
                       TextColor="Yellow" />
                <Label Text="{Binding subtitle}"
                       TextColor="Orange" />
            </StackLayout>
        </ViewCell>
    </DataTemplate>
</ListView.ItemTemplate>
```

Пользовательская ячейка вложена в шаблон DataTemplate, который находится внутри объекта ListView.ItemTemplate. Пользовательская ячейка является производной от класса ViewCell. В состав ячейки входит макет StackLayout с горизонтальной ориентацией и настроенным цветом фона. Внутри макета находятся два элемента управления Label, в которых настроены значения свойств Text и TextColor.

Элемент управления ListView заполняется элементами списка за счет присвоения коллекции свойству ItemsSource. Самым простым способом заполнения элементов списка в элементе управления ListView является использование массива строк. По умолчанию ListView будет вызывать метод ToString и отображать текст в ячейках TextCell, находящихся в каждой строке списка. Далее приведен фрагмент программного кода XAML с примером формирования элемента управления ListView в виде списка строк.

```
<ListView>
    <ListView.ItemsSource>
      <x:Array Type="{x:Type x:String}">
        <x:String>mono</x:String>
        <x:String>monodroid</x:String>
        <x:String>monotouch</x:String>
      </x:Array>
    </ListView.ItemsSource>
</ListView>
```

Такие же функции выполняет программный код Си Шарп, приведенный далее.

```
var listView = new ListView();
listView.ItemsSource = new string[]
{
  "mono",
  "monodroid",
  "monotouch"
};
```

Поскольку свойству ItemsSource был присвоен фиксированный массив строк, то отображаемый в элементе управления список не будет обновляться при добавлении, удалении и изменении элементов списка.

Если необходимо, чтобы список, отображаемый в элементе управления ListView, автоматически изменялся при добавлении, удалении и изменении элементов списка, необходимо использовать динамическую коллекцию данных ObservableCollection, которая выдает уведомления при добавлении и удалении элементов, а также при их обновлении. Пример использования динамической коллекции данных приведен далее в программном коде Си Шарп.

```
ObservableCollection<A> Collection =
                                    new ObservableCollection<A>();
listView.ItemsSource = Collection;

//Г-н Иванов И.И. будет добавлен
//к элементу управления ListView
//так как используется ObservableCollection

Collection.Add(new A(){DisplayName="Г-н Иванов И.И."});
```

Свойства ячеек в списке, отображаемом в элементе управления ListView, можно привязать к свойствам объектов, которые соответствуют свойству ItemsSource элемента управления ListView.

Далее приведен пример программного кода Си Шарп для привязки свойства элемента списка, отображаемого в ListView.

Сначала объявляется класс A, используемый для создания элемента списка.

```
public class A
{
   public string DisplayName {get; set;}
}
```

Далее создается динамическая коллекция данных B типа ObservableCollection, содержащая элементы, созданные с использованием класса A.

Затем данная коллекция присваивается свойству ItemsSource элемента управления ListView с именем AView.

```
ObservableCollection<A> B
            = new ObservableCollection<A>();
public ObservableCollection<A> B
                            {get { return B;}}
public AListPage()
{AView.ItemsSource = B;

   B.Add(new A{ DisplayName="RR"});
   B.Add(new A{ DisplayName="WW"});
   B.Add(new A{ DisplayName="CC"});
}
```

Далее приведен фрагмент программного кода XAML, в котором используется созданный выше элемент управления ListView с именем AView. При этом источником данных с помощью свойства ItemsSource назначена рассмотренная ранее коллекция данных B.

Шаблон для одинакового отображения каждой строки в списке задается далее в программном коде XAML с помощью ListView.ItemTemplate.

```
<ListView x:Name="AView"
          ItemsSource="{Binding B}">
   <ListView.ItemTemplate>
    <DataTemplate>
     <TextCell Text="{Binding DisplayName}" />
    </DataTemplate>
   </ListView.ItemTemplate>
</ListView>
```

Привязка свойств нескольких элементов управления осуществляет синхронизацию в программном коде значений тех свойств элементов управления, которые были привязаны друг к другу. Это упрощает разработку программного кода, так как в случае привязки свойств отпадает необходимость создания обработчиков событий: при изменении значений связанного свойства одного элемента управления измененные значения свойства автоматически присваивается связанному свойству другого элемента управления.

Большие коллекции данных могут стать неудобными для взаимодействия с пользователем, если они представлены с помощью списка, прокрутка которого потребует много времени. Поэтому для улучшения взаимодействия с пользователями может быть использование группирования элементов списка.

Если производится группирование элементов списка для ListView, то для каждой группы добавляется строка заголовка. Для создания группирования элементов списка в элементе управления ListView необходимо:

создать класс для группы, являющейся списком элементов;

создать список групп;

присвоить список свойству ItemsSource элемента управления ListView;

присвоить значение true свойству IsGroupingEnabled;

задать значение свойству GroupDisplayBinding для привязки к тому свойству группы, которое используется в качестве заголовка группы;

задать значение свойству GroupShortNameBinding для привязки к свойству групп, которое используется в качестве краткого имени группы.

Далее приведен фрагмент программного кода Си Шарп, в котором задан класс ElementModel, каждый экземпляр которого будет являться элементом группы. Класс имеет свойства Name и Comment строкового типа.

```
public class ElementModel
{
    public string Name { get; set; }
    public string Comment { get; set; }
    public ElementModel ()
    { ….. }
}
```

Далее в программном коде Си Шарп задается класс группы GroupedElementModel, являющийся дочерним от класса динамической коллекции данных с элементами типа ElementModel. Значения свойств LongName и ShortName предназначены будут использоваться для отображения заголовков групп.

```
public class GroupedElementModel:
            ObservableCollection<ElementModel>
{
    public string LongName { get; set; }
    public string ShortName { get; set; }
}
```

Затем во фрагменте программного кода Си Шарп объявляется и создается коллекция данных grouped, состоящая из списка групп, полученных с помощью класса GroupedElementModel. С помощью метода Add в группу G1, являющуюся экземпляром класса GroupedElementModel, добавляются два элемента, образованные с помощью класса ElementModel. Группа G1 с помощью метода Add добавляется в коллекцию данных grouped. Далее коллекция данных присваивается свойству ItemsSource элемента управления ListView с именем lstView.

```
private ObservableCollection<GroupedElementModel>
                                    grouped { get; set; }
public GroupedList ()
{
grouped =
        new ObservableCollection<GroupedElementModel > ();
var G1 = new GroupedElementModel()
            { LongName = "Group_1", ShortName="Gr_1" };
G1.Add (new ElementModel()
{ Name = "Element_1", Comment =
                        "Первый элемент в группе G1" });
G1.Add (new ElementModel()
{ Name = "Element_2", Comment =
                        "Второй элемент в группе G1" });
}
grouped.Add (G1); lstView.ItemsSource = grouped;
```

В программном коде XAML, приведенном далее, происходит присвоение элементу управления имени lstView. После этого, с помощью присвоения свойству IsGroupingEnabled, значения true делается возможным группирование данных в элементе управления ListView. Далее свойство GroupDisplayBinding привязывается к свойству LongName, которое было определено ранее в классе GroupedElementModel.

Свойство GroupShortNameBinding привязывается к свойству ShortName, которое также было определено ранее в классе GroupedElementModel.

Свойству ItemTemplate элемента управления ListView присваивается шаблон DataTemplate, который с помощью ячейки типа TextCell задает единый внешний вид каждой строки списка. При этом свойства Text и Detail ячейки привязаны соответственно к свойствам Name и Comment, которые были ранее определены в классе ElementModel.

```
<ContentPage.Content>
    <ListView x:Name ="lstView"
              IsGroupingEnabled="true"
              GroupDisplayBinding="{Binding LongName}"
              GroupShortNameBinding=
                                  "{Binding ShortName}">
          <ListView.ItemTemplate>
              <DataTemplate>
                 <TextCell Text="{Binding Name}"
                                Detail =
                               "{Binding Comment}" />
              </DataTemplate>
          </ListView.ItemTemplate>
     </ListView>
</ContentPage.Content>
```

Элемент управления ListView может отображать верхний Header и нижний Footer колонтитулы, которые прокручивается вместе с элементами списка. Для задания колонтитулов используются следующие свойства:

свойства Header и Footer задают текст, размещаемый в колонтитуле, или макет, используемый для формирования колонтитула;

свойства HeaderTemplate и FooterTemplate задают шаблон для формирования колонтитула с поддержкой привязки данных.

Далее приведен фрагмент программного кода XAML, в котором с задаются верхний и нижний колонтитулы в виде строки.

```
<ListView Header="Header"
          Footer= "Footer">
  ...
</ListView>
```

Также приводится фрагмент программного кода, в котором представлены настраиваемые верхний и нижний колонтитулы в виде элементов управления Label. В верхнем колонтитуле задан оливковый цвет текста. В нижнем колонтитуле задан серый цвет текста.

```
<ListView.Header>
  <StackLayout Orientation="Horizontal">
    <Label Text="Header"
        TextColor="Olive"/>
  </StackLayout>
</ListView.Header>
<ListView.Footer>
```

```
    <StackLayout Orientation="Horizontal">
      <Label Text="Footer"
          TextColor="Gray"/>
    </StackLayout>
  </ListView.Footer>
```

Элемент управления ListView имеет свойства HorizontalScrollBarVisibility и VerticalScrollBarVisibility, которые определяют отображение горизонтальной или вертикальной полос прокрутки. Оба свойства могут принимать следующие значения:

значение Default является значением по умолчанию и указывает на то, что поведение полосы прокрутки соответствует настройкам операционной системы и мобильного устройства, на котором запущено программное приложение;

значение Always указывает на то, что полосы прокрутки будут видимы даже в том случае, если вся коллекция в элементе управления ListView отображается полностью;

значение Never указывает на то, что полосы прокрутки не будут видны даже в том случае, если вся коллекция в элементе управления ListView не отображается полностью.

По умолчанию между элементами списка в элементе управления ListView отображаются разделительные линии. Цвет разделительной линии задается с помощью свойства SeparatorColor, которое может принимать следующие значения:

значение Default является значением по умолчанию и соответствует отображению разделительных линий между элементами списка;

значение None соответствует отсутствию разделительных линий между элементами списка на всех платформах.

По умолчанию все строки в отображаемом списке имеют одинаковую высоту. При этом элемент управления ListView имеет два свойства для управления высотой строк:

свойство HasUnevenRows имеет по умолчанию значение false, а при значении true указывает на то, что строки в списке имеют различную высоту;

свойство RowHeight задает высоту каждой строки в случае, если свойство HasUnevenRows имеет значение false.

Если необходимо, чтобы строки в списке имели разную высоту, то для свойства HasUnevenRows задается значение true. При этом высоту строк в этом случае не нужно задавать, так как значения высоты строк будут вычисляться автоматически.

Размеры строки могут быть изменены в ходе работы программного приложения. Для этого используется метод ForceUpdateSize, который обновляет размер ячейки, соответствующей отображаемой строке коллекции.

Далее приведен фрагмент программного кода Си Шарп, в котором приведен пример использования метода ForceUpdateSize в ответ на нажатие на

элемент управления Image. Происходит обновление размера ячейки, и происходит увеличение размеров элемента управления Image.

```
void OnImageTapped (object sender, EventArgs args)
{
   var image = sender as Image;
   var viewCell = image.Parent.Parent as ViewCell;
   if (image.HeightRequest < 250) {
      image.HeightRequest = image.Height + 100;
      viewCell.ForceUpdateSize (); }
}
```

Выбор элементов списка, отображаемого в элементе управления ListView производится с помощью присвоения значения свойству SelectionMode:

значение Single является значением по умолчанию и указывает на то, что можно выбрать в списке один элемент;

значение None указывает на то, что элементы в списке не могут быть выбраны.

Когда пользователь прикасается к элементу в списке, возникают два события:

событие ItemSelected возникает выборе нового элемента;

событие ItemTapped возникает при касании элемента.

Если коснуться одного и того же элемента списка дважды, то будет возникать два раза событие ItemTapped, но при этом событие ItemSelected возникнет только один раз.

В случае возникновения событий ItemSelected и ItemTapped свойству SelectedItem будет присвоено значение выбранного элемента списка. Если значение свойства SelectionMode равно None, то событие ItemSelected не возникает, а свойству SelectedItem присваивается значение null. Но при этом по-прежнему будут возникать событие ItemTapped.

Элементу управления ListView также соответствует событие Scrolled, которое возникает в случае завершения прокрутки.

Для демонстрации возможностей мобильного программного приложения по работе со списками и коллекциями с помощью элемента управления ListView создадим проект ListViewDemo для мультиплатформенного программного приложения Xamarin.Forms с использованием шаблона «Пустой».

После создания проекта необходимо зайти в диалоговое окно «Обозреватель решений», выделить и открыть проект ListViewDemo и далее с помощью рассмотренной ранее последовательности действий создать классы RocketModel и GroupedRocketModel.

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

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;

namespace ListViewDemo
{
   public class RocketModel
   {       public string Name {get; set;}
           public string Country {get; set;}
   }
}
```

В программном коде объявляется класс RocketModel для объявления элемента коллекции. Каждый элемент коллекции имеет свойства Name и Country строкового типа.

Далее в диалоговом окне «Обозреватель решений» необходимо открыть файл GroupedRocketModel.cs и набрать программный код, приведенный далее.

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;

namespace ListViewDemo
{
public class GroupedRocketModel:
                          ObservableCollection<RocketModel>
   {
      public string GroupName {get; set;}
      public string PayLoad {get; set;}
   }
}
```

В программном коде объявляется класс GroupedRocketModel для создания коллекции. Класс является дочерним классом от класса динамической коллекции данных ObservableCollection с элементами типа RocketModel. На классы RocketModel и GroupedRocketModel будут производиться ссылки из программного кода в файлах MainPage.xaml и MainPage.xaml.cs.

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

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="ListViewDemo.MainPage">
<ContentPage.Content>
   <ListView x:Name="lstView"
```

```
            RowHeight="70"
            IsGroupingEnabled="true"
            ItemSelected="OnSelection">
        <ListView.ItemTemplate>
          <DataTemplate>
              <ViewCell>
                <StackLayout BackgroundColor="Orange">
                    <Label Text="{Binding Name}"
                        TextColor="Black"
                        FontAttributes="Bold"
                        FontSize="20"
                        HorizontalTextAlignment="Center"
                        VerticalTextAlignment="Center"/>
                    <Label Text="{Binding Country}"
                        TextColor="Black"
                        FontAttributes="Bold"
                        FontSize="20"
                        HorizontalTextAlignment="Center"
                        VerticalTextAlignment="Center"/>
                </StackLayout>
              </ViewCell>
          </DataTemplate>
        </ListView.ItemTemplate>
        <ListView.GroupHeaderTemplate>
            <DataTemplate>
              <ViewCell>
                <StackLayout BackgroundColor="Red">
                  <Label Text="{Binding GroupName}"
                        TextColor="Black"
                        FontAttributes="Bold"
                        FontSize="22"
                        HorizontalTextAlignment="Center"/>
                  <Label Text="{Binding PayLoad}"
                        TextColor="Black"
                        FontAttributes="Bold"
                        FontSize="22"
                        HorizontalTextAlignment="Center"/>
                </StackLayout>
              </ViewCell>
            </DataTemplate>
          </ListView.GroupHeaderTemplate>
      </ListView>
</ContentPage.Content>
</ContentPage>
```

В программном коде в странице типа ContentPage размещается элемент управления ListView с именем lstView. В элементе управления предусматривается возможность группирования и назначена высота строк в списке.

Предусмотрена возможность выбора элемента списка с помощью обработки события ItemSelected. Программный код обработчика OnSelection приведен в файле MainPage.xaml.cs.

Для настройки элементов списка применяется пользовательская ячейка типа ViewCell.

Ячейка содержит макет типа StackLayout с вертикальной ориентацией, внутри которого находятся два элемента управления Label, в которых свойства Text привязаны к свойствам Name и Country, объявленным ранее в классе RocketModel.

Для настройки заголовков групп свойству GroupHeaderTemplate присваивается шаблон в виде ячейки типа ViewCell.

Ячейка содержит макет типа StackLayout с вертикальной ориентацией, внутри которого находятся два элемента управления Label, в которых свойства Text привязаны к свойствам GroupName и PayLoad, объявленным ранее в классе GroupedRocketModel.

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

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using Xamarin.Forms;
namespace ListViewDemo
{
   public partial class MainPage: ContentPage
   {
   private ObservableCollection<GroupedRocketModel>
                                          grouped {get; set;}
            public MainPage()
            {
                InitializeComponent();
                grouped =
             new ObservableCollection<GroupedRocketModel>();
                 var Type1 = new GroupedRocketModel()
                    {
              GroupName = "Ракеты-носители тяжелого класса",
              PayLoad = "Полезная нагрузка 20-28 тонн "
                    };
                 var Type2 = new GroupedRocketModel()
                    {
              GroupName = "Ракеты-носители среднего класса",
```

```
PayLoad = "Полезная нагрузка 5-20 тонн"
        };
    var Type3 = new GroupedRocketModel()
        {
GroupName = "Ракеты-носители легкого класса",
PayLoad = "Полезная нагрузка до 5 тонн"
        };
        Type1.Add(new RocketModel()
        {Name = "Протон",
         Country = "Россия"});
        Type1.Add(new RocketModel()
        {Name = "Дельта IV Heavy",
         Country = "США"});
        Type1.Add(new RocketModel()
        {Name = "Чанчжэн-5",
          Country = "Китай"});
        Type1.Add(new RocketModel()
        {Name = "Ариан-5 ES",
         Country = "Евросоюз"});
         Type1.Add(new RocketModel()
        {Name = "Falcon 9 FT",
          Country = "США"});
         Type2.Add(new RocketModel()
         {Name = "Союз-2", Country = "Россия"});
         Type2.Add(new RocketModel()
        {Name = "Зенит-3SLBF",
        Country = "Украина"});
        Type2.Add(new RocketModel()
        {Name = "Чанчжэн 3B",
         Country = "Китай"});
        Type2.Add(new RocketModel()
        {Name = "Delta-II",
         Country = "США"});
         Type2.Add(new RocketModel()
         {Name = "H-II",
          Country = "Япония"});
         Type3.Add(new RocketModel()
         {Name = "Minotaur",
          Country = "США"});
         Type3.Add(new RocketModel()
         {Name = "Космос-3М",
          Country = "Россия"});
         Type3.Add(new RocketModel()
         {Name = "PSLV-XL",
          Country = "Индия"});
```

```
                    Type3.Add(new RocketModel()
                    {Name = "Эпсилон",
                     Country = "Япония"});
                    Type3.Add(new RocketModel()
                    {Name = "Чанчжэн-11",
                     Country = "Китай"});
                     grouped.Add(Type1);
                     grouped.Add(Type2);
                     grouped.Add(Type3);
                    lstView.ItemsSource = grouped;
            }
            void OnSelection
                  (object sender, SelectedItemChangedEventArgs e)
            {
                  RocketModel rm;
                  String all;
                  if (e.SelectedItem == null)
                  {return;}
                  rm = (RocketModel)e.SelectedItem;
                  all = rm.Name + " (" + rm.Country+ ")";
                 DisplayAlert("Выбран элемент", all, "Ok");
            }
      }
}
```

В программном коде сначала объявляется свойство grouped, представляющее собой динамическую коллекцию данных ObservableCollection с элементами типа GroupedRocketModel.

Свойству grouped сначала присваивается пустая коллекция. Затем производится формирование трех групп Type1, Type2 и Type3 и присвоение значений их свойствам GroupName и PayLoad.

После этого производится добавление элементов в группы с помощью метода Add. Каждый элемент является экземпляром класса RocketModel и имеет свойства Name и Country.

Сформированные группы с помощью метода Add добавляются в коллекцию grouped, которая присваивается свойству ItemsSource элемента управления ListView с именем lstView.

Приведен обработчик OnSelection для события ItemSelected. Если элемент списка выбран, то выбранный элемент передается в обработчик в качестве значения свойства SelectedItem параметра e типа SelectedItemChangedEventArgs.

Из элемента, присвоенного свойству SelectedItem, с помощью переменной rm извлекаются значения свойств Name и Country.

С использованием метода DisplayAlert производится отображение диалогового окна, в котором с помощью переменной all выдается информация, содержащаяся в свойствах Name и Country.

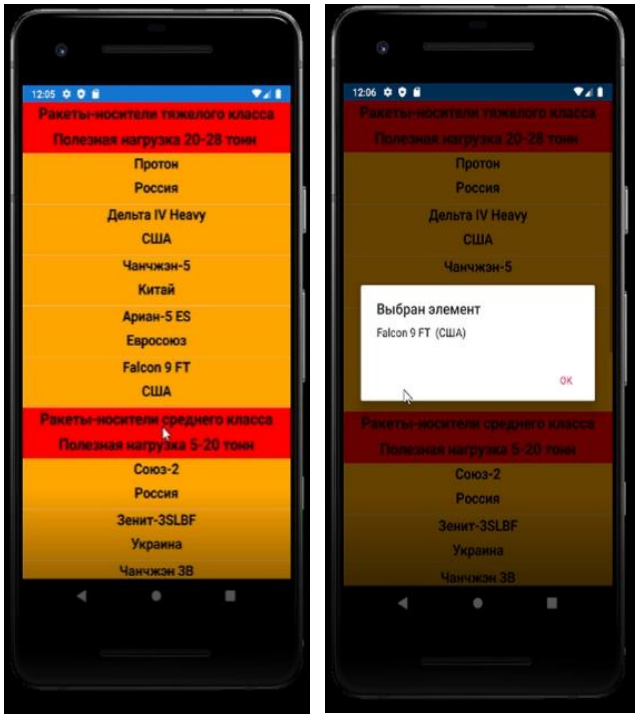
Для демонстрации возможностей мобильных программных приложений по работе со списками и коллекциями с помощью элемента управления ListView необходимо сначала собрать решение, а затем, в случае отсутствия ошибок, запустить программное приложение.

После запуска программного приложения отображается коллекция, состоящая из групп, заголовки которых выделены красным фоном (рис. 120).

Каждая группа состоит из списка элементов, с фоном оранжевого цвета. Коллекция может прокручивается вверх и вниз.

При выборе любого элемента списка возникает событие ItemSelected. При возникновении данного события срабатывает обработчик, который отображает диалоговое окно с названием выбранного элемента.

Для закрытия диалогового окна необходимо нажать кнопку «ОК» в диалоговом окне.



Далее рассматриваются возможности мобильных программных приложений по работе со списками и коллекциями с помощью элемента управления TableView.

Данный элемент управления предназначен для отображения прокручиваемых списков данных, в которых строки не используют общий шаблон внешнего вида.

В элементе управления TableView для задания источника коллекции не используется свойство ItemsSource.

Элементы списка добавляются в коллекцию в качестве дочерних элементов и распределены по секциям.

Корневой секцией элемента управления TableView является секция TableRoot. Данная секция является родительской для одной или нескольких секций TableSection.

При этом в состав секции входит заголовок и один или несколько экземпляров ячеек.

Далее приведен фрагмент программного кода XAML, в котором рассматривается пример создания двух ячеек типа SwitchCell, в одной из которых переключатель находится во включенном состоянии.

```
<TableView Intent="Settings">
  <TableRoot>
    <TableSection Title="Services">
      <SwitchCell Text="New Voice Mail" />
      <SwitchCell Text="New Mail" On="true" />
    </TableSection>
  </TableRoot>
</TableView>
```

Элемент управления TableView имеет свойство Intent, которое задает режим использования элемента управления TableView, а также свойство TextColor для задания цвета текста в заголовке секции TableSection. Свойство Intent может принимать следующие значения:

значение Data свидетельствует об использовании элемента управления TableView для отображения и прокрутки коллекции в виде списка данных (при этом лучшим вариантом для отображения и прокрутки списков данных является элемент управления ListView);

значение Form свидетельствует об отображении элемента управления TableView в виде формы для ввода данных;

значение Menu свидетельствует об использовании элемента управления TableView для вывода меню;

значение Settings свидетельствует об использовании элемента управления TableView для отображения списка параметров конфигурации.

Элемент управления TableView может использовать такие же встроенные ячейки, как и элемент управления ListView. При этом для работы с TableView наиболее часто используются ячейки SwitchCell и EntryCell [79]. Подробное описание свойств встроенных ячеек приведено ранее при рассмотрении работы с элементом управления ListView. Если возможностей, предоставляемых встроенными ячейками, недостаточно, то используются пользовательские ячейки.

Далее приведен фрагмент программного кода XAML, в котором рассматривается пример пользовательской ячейки, являющейся экземпляром класса ViewCell и содержащей макет типа StackLayout с горизонтальным размещением находящихся внутри него элементов управления Image и Label. При этом, ячейка находится внутри раздела TableSection.

```
<TableSection Title="Rocket">
```

```
        <ViewCell>
            <StackLayout Orientation="Horizontal">
                    <Image Source="foto.jpg" />
                    <Label Text="Название"
                           TextColor="Red" />
            </StackLayout>
        </ViewCell>
    </TableSection>
```

Элемент управления TableView имеет два свойства, которые можно использовать для задания высоты строк:

свойство RowHeight задает высоту каждой строки в элементе управления TableView;

свойство HasUnevenRows имеет по умолчанию значение false, что соответствует одинаковой высоте строк в элементе управления TableView, при этом значении true соответствует различному значению высоты строк.

Следует учесть, что в случае, если значение HasUnevenRows равно true высота строк в элементе управления TableView вычисляется автоматически без задания значения свойства RowHeight.

Высота ячейки в TableView автоматически обновляется для операционной системы Android. При этом для операционной системы iOS необходимо принудительно обновлять высоту строки, задав значение true для свойства HasUnevenRows и вызвав метод ForceUpdateSize.

Далее приведен фрагмент программного кода XAML, в котором рассмотрено использование метода ForceUpdateSize. Сначала создается ячейка с именем vc, которая меняет состояние в случае касания к ней и появления события Tapped. В начальный момент времени элемент управления Label является невидимым.

Ячейка содержит два элемент управления, один из которых имеет имя target. В случае нажатия на ячейку происходит обращение к обработчику OnViewCellTapped.

```
    <TableView
          HasUnevenRows="true">
      <TableRoot>
        ...
        <TableSection ...>
        ...
          <ViewCell x:Name="vc"
                Tapped="OnViewCellTapped">
            <Grid Margin="15,0">
              <Grid.RowDefinitions>
                <RowDefinition Height="Auto" />
                <RowDefinition Height="Auto" />
              </Grid.RowDefinitions>
              <Label Text="Нажать на ячейку" />
```

```
                <Label x:Name="target"
                    Grid.Row="1"
                    Text="Ячейка изменяет размер"
                    IsVisible="false" />
            </Grid>
        </ViewCell>
    </TableSection>
  </TableRoot>
</TableView>
```

В приведенном далее фрагменте программного кода Си Шарп показано, что при нажатии на ячейку обработчик OnViewCellTapped значение свойства IsVisible элемента управления Label с именем target изменяется на противоположное.

Затем с помощью метода ForceUpdateSize производится обновление ячейки с именем vc.

В результате при видимом состоянии обоих элементов управления Label высота ячейки ViewCell будет больше, чем при невидимом состоянии элемента управления с именем target.

```
void OnViewCellTapped (object sender, EventArgs e)
{  target.IsVisible = !target.IsVisible;
   vc.ForceUpdateSize();
}
```

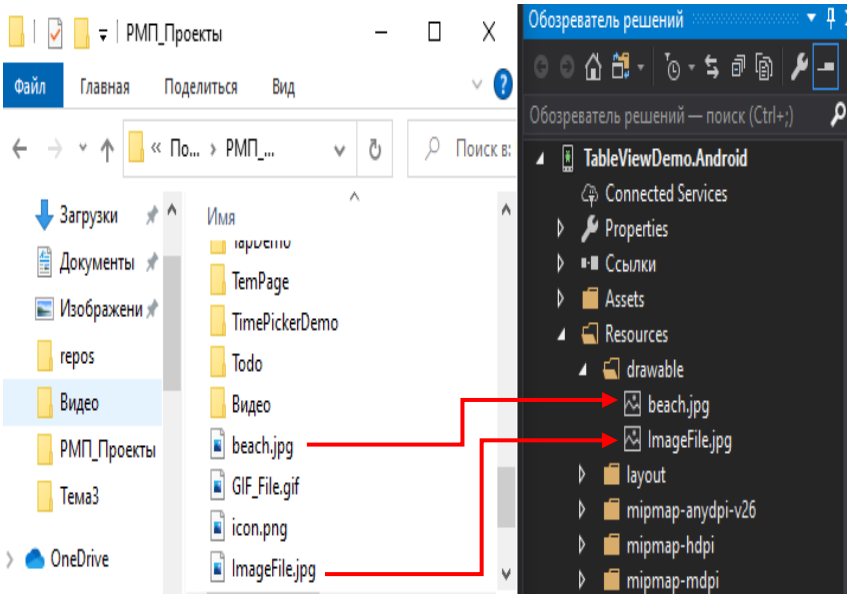
Для демонстрации возможностей мобильного программного приложения по работе со списками и коллекциями с помощью элемента управления TableView создается проект TableViewDemo для мультиплатформенного программного приложения Xamarin.Forms с использованием шаблона «Пустой».

После создания проекта необходимо загрузить в него изображения для обеспечения работы с ячейками ImageCell, с которыми будет производиться работа в проекте.

Для этого необходимо зайти в диалоговое окно «Обозреватель решений», выделить и открыть проект TableViewDemo.Android, выделить и открыть папку Resourcew и далее загрузить два файла с изображениями в папку Drawable (рис. 121). В диалоговом окне «Обозреватель решений» выделить и открыть файл MainPage.xaml и в нем набрать программный код XAML, приведенный далее.

В программном коде страница типа ContentPage содержит элемент управления TableView, для которого с помощью свойства Intent установлен режим отображения данных.

Также с помощью свойства HasUnevenRows сделана возможной различная высота ячеек.



В элементе управления TableView размещены две секции. В первой секции четыре строки. Каждой из строк соответствует один из типов встроенных ячеек (ImageCell, SwitchCell, EntryCell, TextCell). В ячейке ImageCell заданы основной и вспомогательный тексты, цвет основного и вспомогательного текста, а также источник изображения для ячейки. В ячейке SwitchCell задан основной текст, состояние, а также цвет элемента управления во включенном состоянии. В ячейке EntryCell задан текст, отображаемый слева от поля ввода текста, и его цвет, а также текст внутри ячейки. В ячейке TextCell заданы основной и вспомогательный тексты, а также их цвет. Во второй секции пять строк. Первым четырем строкам также соответствуют встроенные ячейки (ImageCell, SwitchCell, EntryCell, TextCell), для которых заданы значения тех же свойств, что и для ячеек в первой секции. В пятой строке второй секции размещается пользовательская ячейка ViewCell с именем vc, в которой располагается макет Grid. В макете Grid создаются две строки, в каждой из которых располагается элемент управления Label, один из которых имеет имя targ. Пользовательская ячейка будет реагировать на нажатия, которым соответствует событие Tapped. Для реагирования на возникновение этого события производится обращение к обработчику OnViewCellTapped, программный код которого приведен в файле MainPage.xaml.cs.

В ячейки ImageCell загружаются изображения из файлов beach.jpg и ImageFile.jpg, которые были ранее размещены в папке Drawable проекта TableViewDemo.Android.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="TableViewDemo.MainPage">
  <TableView Intent="Data"
        HasUnevenRows="true"
        BackgroundColor="LightYellow">
```

```
<TableRoot>
  <TableSection Title="Первая секция TableView"
          TextColor="Black">
    <ImageCell Text="ImageCell в 1-й секции"
          Detail="Xamarin"
          TextColor="Black"
          DetailColor="Black"
          ImageSource="beach.jpg" />
    <SwitchCell Text="SwitchCell в 1-й секции"
          On="True"
          OnColor="Green"/>
    <EntryCell Label="EntryCell в 1-й секции"
          Text="Введите Фамилию"
          LabelColor="Black"/>
    <TextCell Text="TextCell в 1-й секции"
          Detail="Работа с TableView"
          TextColor="Black"
          DetailColor="Black"/>
  </TableSection>
  <TableSection Title="Вторая секция TableView"
          TextColor="Red">
    <ImageCell Text="ImageCell во 2-й секции"
          Detail="Xamarin"
          TextColor="Red"
          DetailColor="Red"
          ImageSource="ImageFile.jpg" />
    <SwitchCell Text="SwitchCell во 2-й секции"
          On="True"
          OnColor="Green"/>
    <EntryCell Label="EntryCell во 2-й секции"
          Text="Введите Имя"
          LabelColor="Red"/>
    <TextCell Text="TextCell"
          Detail="TextCell во 2-й секции"
          TextColor="Red"
          DetailColor="Red"/>
    <ViewCell x:Name="vc"
          Tapped="OnViewCellTapped">
      <Grid Margin="15,0">
        <Grid.RowDefinitions>
          <RowDefinition Height="Auto" />
          <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Label Text="Нажать на ячейку"
            FontSize="16"
```

```
                FontAttributes="Bold"
                BackgroundColor="Orange"/>
            <Label x:Name="targ"
                Grid.Row="1"
              Text="Пользовательская ячейка во 2-й секции"
                IsVisible="false"
                FontSize="16"
                FontAttributes="Bold"
                BackgroundColor="Orange"/>
          </Grid>
        </ViewCell>
      </TableSection>
    </TableRoot>
  </TableView>
</ContentPage>
```

В диалоговом окне «Обозреватель решений» выделить и открыть файл MainPage.xaml.cs и наберем программный код Си Шарп, приведенный далее.

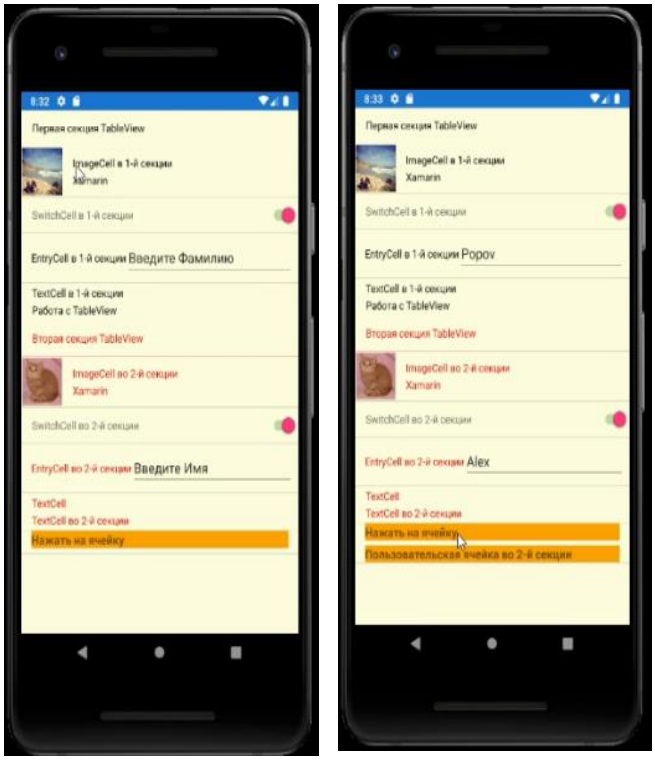
```
using System;
using Xamarin.Forms;
namespace TableViewDemo
{  public partial class MainPage: ContentPage
  {
      public MainPage()
        {
            InitializeComponent();
        }
      void OnViewCellTapped(object sender, EventArgs e)
      {   targ.IsVisible = !targ.IsVisible;
          vc.ForceUpdateSize();
      }
  }
}
```

В программном коде содержится обработчик нажатия на пользовательскую ячейку ViewCell с именем vc.

Обработчик изменяет значение свойства IsVisible для элемента управления Label с именем targ, расположенного в пользовательской ячейке. Производится использование метода ForceUpdateSize для обновления размеров пользовательской ячейки ViewCell, которое происходит вследствие изменения значения свойства IsVisible элемента управления Label.

Для демонстрации возможностей мобильного программного приложения по работе со списками и коллекциями с использованием элемента управления TableView необходимо сначала собрать решение, а затем, в случае отсутствия

ошибок, запустить программного приложение. После запуска программного приложения в пользовательском интерфейсе отображаются две секции.



В первой секции четыре строки. Каждой строке этой секции соответствует один из типов встроенных ячеек. Во второй секции пять строк. Первые четыре строки, как и в первой секции, соответствуют одному из типов встроенных ячеек. В пятой строке второй секции находится встроенная ячейка, которая может изменять высоту. В случае нажатия на эту ячейку происходит изменение ее высоты за счет того, что изменяется видимость одного из элементов управления Label внутри пользовательской ячейки.