Работа с коллекциями в Xamarin.Forms с помощью элемента управления CollectionView

Во время работы с пользовательским интерфейсом часто приходится иметь дело со списками и коллекциями данных различного типа, требующими прокрутки вследствие того, что элементы списков и данные невозможно отобразить в полном объеме в пределах пользовательского интерфейса.

Для прокрутки списков и коллекций могут быть использованы элементы управления пользовательского интерфейса, относящиеся в Xamarin.Forms к визуальным элементам для отображения коллекций (CarouselView, IndicatorView, CollectionView, ListView, TableView).

Перейдем к изучению элемента управления CollectionView, который служит для отображения коллекций данных в пользовательском интерфейсе программного приложения с использованием различных макетов.

Элемент управления CollectionView имеет следующие свойства, определяющие его внешний вид и отображаемые в нем данные:

свойство ItemsSource имеет значение по умолчанию значение null и задает коллекцию элементов для отображения;

свойство ItemTemplate задает шаблон, применяемый к каждому элементу отображаемой коллекции.

Элемент управления CollectionView заполняется данными за счет присваивания его свойству ItemsSource коллекции данных. По умолчанию элементы коллекции отображаются в элементе управления CollectionView в вертикальном списке. В XAML это достигается с помощью расширения разметки Binding. В Си Шарп свойство ItemsSource привязывается к коллекции элементов данных с помощью метода SetBinding.

XAML

```
<CollectionView ItemsSource="{Binding Rockets}" />
```

Си Шарп

```
CollectionView collectionView = new CollectionView();
collectionView.SetBinding(ItemsView.ItemsSourceProperty, " Rockets ");
```

Внешний вид каждого элемента коллекции в элементе управления CollectionView может быть определен путем присвоения его свойству ItemTemplate значения шаблона DataTemplate.

Далее приведен фрагмент программного кода, в котором в шаблоне DataTemplate находится макет Frame, содержащий элемент управления Image, который привязан к свойству ImageUrl элементов коллекции:

```
<CollectionView.ItemTemplate>
  <DataTemplate>
     <Frame BorderColor="Violet"/>
        <Image Source="{Binding ImageUrl}"
               Aspect="AspectFill"/>
     </Frame>
```

```
    </DataTemplate>
  </CollectionView.ItemTemplate>
```

Внешний вид каждого элемента коллекции также можно выбрать, задав для свойства ItemTemplate объект типа DataTemplateSelector, который позволит выбирать шаблоны элементов.

Далее приведен фрагмент программного кода, в котором определены два шаблона DataTemplate, а также объект RSelector типа DataTemplateSelector.

```
<ContentPage.Resources>
    <DataTemplate x:Key="Template1">
      ...
    </DataTemplate>
    <DataTemplate x:Key="Template2">
      ...
    </DataTemplate>
    <controls: RSelector x:Key="Sel"
        T1="{StaticResource Template1}"
        T2="{StaticResource Template2}" />
  </ContentPage.Resources>
  <CollectionView ItemsSource="{Binding Rockets}"
            ItemTemplate="{StaticResource Sel}" />
</ContentPage>
```

В результате работы селектора к элементу коллекции с помощью метода OnSelectTemplate применяется шаблон T1 в случае, если имя элемента Rocket содержит строку "Tactical".

Если имя элемента не содержит строки "Tactical", то метод OnSelectTemplate применяет к элементу шаблон T2

```
public class RSelector: DataTemplateSelector
{  public DataTemplate T1 { get; set; }
  public DataTemplate T2 { get; set; }
  protected override DataTemplate OnSelectTemplate
        (object item, BindableObject container)
   {     return ((Rocket)item).
        Location.Contains("Tactical") ? T1 : T2;
   }
}
```

Элемент управления CollectionView имеет следующие свойства для задания макета, с помощью которого элемент управления отображает коллекцию элементов данных:

свойство ItemsLayout задает используемый макет;

свойство ItemSizingStrategy определяет стратегию задания размеров элементов коллекции.

По умолчанию элемент управления CollectionView отображается в виде вертикального списка.

Но можно использовать любой из следующих макетов:

макет VerticalList задает список из одного столбца, который увеличивается по вертикали при добавлении новых элементов в коллекцию;

макет HorizontalList задает горизонтальный список, который увеличивается горизонтально по мере добавления новых элементов в коллекцию;

макет VerticalGrid задает сетку с несколькими столбцами, которая растет вертикально по мере добавления новых элементов в коллекцию;

макет HorizontalGrid задает многострочную сетку, которая растет горизонтально по мере добавления новых элементов в коллекцию.

Свойство ItemSizingStrategy может иметь следующие значения:

значение MeasureAllItems является значением по умолчанию, которое указывает на то, что размеры задаются для каждого элемента коллекции по отдельности;

значение MeasureFirstItem указывает на то, что задаются размеры только первого элемента коллекции, а все остальные элементы коллекции будут иметь такие же размеры, как и первый элемент.

Каждый макет представляет собой объект типа ItemsLayout, который имеет следующие свойства:

свойство Orientation имеет тип ItemsLayoutOrientation и задает направление, в котором в элементе управления CollectionView будет расширяться отображаемая коллекция при добавлении в нее новых элементов;

свойство SnapPointsAlignment, задает способ выравнивания точек привязки элементов коллекции;

свойство SnapPointsType задает поведение точек привязки элементов при прокрутке коллекции.

Свойство Orientation типа ItemsLayoutOrientation может принимать следующие значения:

значение Vertical указывает, что элемент управления будет расширяться по вертикали при добавлении элементов в коллекцию;

значение Horizontal указывает, что элемент управления CollectionView будет расширяться по горизонтали при добавлении элементов в коллекцию.

В случае, если в элементе управления CollectionView коллекция отображается с помощью макетов в виде горизонтальных или вертикальных списков, используется класс LinearItemsLayout, являющийся дочерним от класса ItemsLayout.

Данный класс имеет свойство ItemSpacing, которое задает пустое пространство вокруг каждого элемента коллекции. Значение этого свойства по умолчанию равно 0, а его значение всегда должно быть больше или равно 0.

При отображении коллекции в виде списка для класса LinearItemsLayout можно задать значения Vertical или Horizontal, которые соответствуют вертикальному или горизонтальному списку.

Кроме того, для создания списков для класса LinearItemsLayout можно создать объект ItemsLayoutOrientation и присвоить этому объекту значение Vertical или Horizontal.

В случае, если в элементе управления CollectionView коллекция отображается с помощью сетки, используется класс GridItemsLayout, являющийся дочерним от класса ItemsLayout. Класс GridItemsLayout имеет следующие свойства:

свойство VerticalItemSpacing задает пустое пространство вокруг каждого элемента коллекции по вертикали, при этом значение этого свойства по умолчанию равно 0, а его значение всегда должно быть больше или равно 0;

свойство HorizontalItemSpacing задает пустое пространство вокруг каждого элемента коллекции по горизонтали, при этом значение этого свойства также по умолчанию равно 0, а его значение также должно быть больше или равно 0;

свойство Span задает число столбцов или строк, отображаемых в сетке, при этом значение этого свойства по умолчанию равно 1, а его значение всегда должно быть больше или равно 1.

При использовании макета LinearItemsLayout в виде списка элемент управления CollectionView по умолчанию отображает коллекцию элементов в виде вертикального списка.

Далее приведены фрагменты программного кода, в которых показано, что для отображения элементов коллекции в виде горизонтального списка в XAML необходимо присвоить свойству ItemsLayout значение HorizontalList. Также можно задать для свойства ItemsLayout объект LinearItemsLayout и для свойства Orientation этого объекта задать значение Horizontal.

```
<CollectionView ItemsSource="{Binding Rockets}"
                ItemsLayout="HorizontalList">
  …
</CollectionView>

<CollectionView ItemsSource="{Binding Rockets}">
  <CollectionView.ItemsLayout>
    <LinearItemsLayout Orientation="Horizontal"/>
  </CollectionView.ItemsLayout>
  …
</CollectionView>
```

При использовании языка Си Шарп для задания горизонтального списка свойству ItemsLayout присваивается значение Horizontal объекта LinearItemsLayout.

```
CollectionView collectionView = new CollectionView
{ ...
  ItemsLayout = LinearItemsLayout.Horizontal
};
```

При использовании макета GridItemsLayout в виде сетки элемент управления CollectionView по умолчанию отображает коллекцию элементов в виде одного столбца по вертикали.

Далее приведены фрагменты программного кода, в которых показано, что для отображения элементов коллекции в горизонтальной сетке в XAML необходимо присвоить свойству ItemsLayout значение HorizontalGrid.

```
<CollectionView ItemsSource="{Binding Rockets}"
          ItemsLayout="HorizontalGrid, 4">
  <CollectionView.ItemTemplate>
    <DataTemplate>
     …
    </DataTemplate>
  </CollectionView.ItemTemplate>
</CollectionView>
```

Также можно задать для свойства ItemsLayout объект GridItemsLayout и для свойства Orientation этого объекта задать значение Horizontal.

По умолчанию в случае использования макета GridItemsLayout с горизонтальной ориентацией элементы коллекции будут размещаться в одной строке.

Однако в программном коде свойству Span присваивается значение, равное 4. В результате получается сетка из четырех строк, которая будет увеличивается в горизонтальном направлении при добавлении новых элементов в коллекцию.

```
<CollectionView ItemsSource="{Binding Rockets}">
  <CollectionView.ItemsLayout>
    <GridItemsLayout Orientation="Horizontal"
                Span="4" />
  </CollectionView.ItemsLayout>
  ...
</CollectionView>
```

Элемент управления CollectionView может отображать верхний и нижний колонтитулы, которые могут прокручиваться вместе с элементами списка.

Верхний и нижний колонтитулы могут быть строками, элементами управления или шаблонами, заданными с помощью DataTemplate. Для работы с колонтитулами имеются следующие свойства:

свойство Header задает строку, ссылку или элемент управления представление, которые будут отображаться в начале коллекции;

свойство HeaderTemplate типа DataTemplate задает шаблон, используемый для форматирования данных, содержащихся в свойстве Header;

свойство Footer задает строку, ссылку или элемент управления, которые будут отображаться в конце списка;

свойство FooterTemplate типа DataTemplate задает шаблон, используемый для форматирования данных, содержащихся в свойстве Footer.

Далее приводится фрагмент программного кода, в котором для свойств Header и Footer заданы текстовые значения.

```
<CollectionView ItemsSource="{Binding Rockets}"
          Header=" Rockets"       Footer="2021">
  ...
</CollectionView>
```

Также для свойств Header и Footer можно задать значения в виде элементов управления.

Может быть задан один или несколько элементов управления.

Далее приведены фрагменты программного кода, в которых для свойств Header и Footer задано по одному элементу управления Label.

```
<CollectionView.Header>
      <Label Text="Missiles" FontSize="Small"/>
</CollectionView.Header>
<CollectionView.Footer>
      <Label Text="Targets"  FontSize="Large"/>
</CollectionView.Footer>
```

Значения свойств HeaderTemplate и FooterTemplate можно задать с помощью шаблона DataTemplate.

Далее приведен фрагмент программного кода, в котором рассмотрена привязка свойств Header и Footer к имеющимся шаблонам, в качестве которых выступают элементы управления Label.

```
<CollectionView ItemsSource="{Binding Rockets}"
      Header="{Binding .}"  Footer="{Binding .}">
  <CollectionView.HeaderTemplate>
    <DataTemplate>
      <Button Text="Start" FontAttributes="Bold" />
    </DataTemplate>
  </CollectionView.HeaderTemplate>
  <CollectionView.FooterTemplate>
    <DataTemplate>
      <Button Text="Explosion"
       FontAttributes="Bold" />
      </StackLayout>
    </DataTemplate>
  </CollectionView.FooterTemplate>
  ...
```

```
</CollectionView>
```

По умолчанию между элементами коллекции не существует пробела.

Для изменения пробела между элементами коллекции при использовании макета в виде списка производится изменение значения свойства ItemSpacing.

```
<CollectionView ItemsSource="{Binding Rockets}">
  <CollectionView.ItemsLayout>
    <LinearItemsLayout Orientation="Vertical"
                ItemSpacing="20" />
  </CollectionView.ItemsLayout>
  ...
</CollectionView>
```

Для изменения пробела по вертикали и горизонтали между элементами коллекции при использовании макета в виде сетки производится изменение значений свойств VerticalItemSpacing и HorizontalItemSpacing.

```
<CollectionView ItemsSource="{Binding Rockets}">
  <CollectionView.ItemsLayout>
    <GridItemsLayout Orientation="Vertical"
              Span="2"
              VerticalItemSpacing="20"
              HorizontalItemSpacing="30" />
  </CollectionView.ItemsLayout>
  ...
</CollectionView>
```

Элемент управления CollectionView имеет следующие свойства для управления выбором элемента коллекции:

свойство SelectionMode задает режим выбора элемента коллекции;

свойство SelectedItem задает выбранный элемент в списке, при этом значение свойства равно null, если ни один элемент коллекции не выбран;

свойство SelectedItems задает несколько выбранных элементов коллекции, при этом значение свойства равно null, если ни один элемент коллекции не выбран;

свойство SelectionChangedCommand задает команду, которая выполняется при изменении смене элемента коллекции, отображаемого в данный момент времени;

свойство SelectionChangedCommandParameter задает параметр, который передается в команду, указанную в свойстве SelectionChangedCommand.

По умолчанию выбор элементов коллекции в элементе управления CollectionView отключен.

Изменить режим выбора можно с помощью свойства SelectionMode, которое может принимать следующие значения:

значение None является значением по умолчанию и указывает на то, что элементы не могут быть выбраны;

значение Single указывает на то, что можно выбрать один элемент коллекции;

значение Multiple указывает на то, что можно выбрать несколько элементов коллекции.

При изменении значения свойства SelectedItem элемента управления CollectionView происходит событие SelectionChanged.

Кроме того, это событие возникает при изменении значения свойства SelectedItems.

Событию SelectionChanged соответствует объект SelectionChangedEventArgs, который имеет два свойства:

свойство PreviousSelection задает список элементов коллекции, которые были выбраны до осуществления нового выбора;

свойство CurrentSelection задает список элементов коллекции, которые являются выбранными в текущий момент времени.

Кроме того, элемент управления CollectionView имеет метод UpdateSelectedItems, который обновляет значение свойства SelectedItems за счет присвоения списка вновь выбранных элементов коллекции.

Если свойство SelectionMode имеет значение Single, то можно выбрать только один элемент коллекции. Если выбран элемент коллекции, то свойству SelectedItem будет присвоено значение в виде выбранного элемента.

При изменении значения этого свойства выполняется команда SelectionChangedCommand, в которую передается параметр в соответствии со значением свойства SelectionChangedCommandParameter. После этого происходит событие SelectionChanged.

Далее приведен фрагмент программного кода, в котором рассмотрен элемент управления CollectionView, в котором предусмотрен выбор только одного элемента коллекции.

Производится выполнение обработчика OnCollectionViewSelectionChanged при срабатывании события SelectionChanged.

В результате обработчик получает название выбранного ранее элемента коллекции, а также название элемента коллекции, выбранного в текущий момент времени.

```
<CollectionView ItemsSource="{Binding Rockets}"
            SelectionMode="Single"
            SelectionChanged=
                  "OnCollectionViewSelectionChanged">
   ...
</CollectionView>
void OnCollectionViewSelectionChanged
        (object sender, SelectionChangedEventArgs e)
{  string previous =
     (e.PreviousSelection.FirstOrDefault() as Rocket)?.Name;
   string current =
     (e.CurrentSelection.FirstOrDefault() as Rocket)?.Name;
```

```
  ...
}
```

Если свойство SelectionMode имеет значение Multiple, то можно выбрать несколько элементов коллекции. При этом свойству SelectedItems будут присвоены несколько выбранных элементы.

При изменении значения этого свойства также выполняется команда SelectionChangedCommand, в которую передается параметр в соответствии со значением свойства SelectionChangedCommandParameter.

Также происходит событие SelectionChanged. Далее приведен фрагмент программного кода, в котором рассмотрен элемент управления CollectionView, в котором предусмотрен выбор нескольких элементов коллекции.

Выполнение обработчика OnCollectionViewSelectionChanged предусматривает присвоение. В результате обработчик получает список выбранных ранее элементов коллекции, а также список элементов коллекции, выбранных в текущий момент времени

```
<CollectionView ItemsSource="{Binding Rockets}"
          SelectionMode="Multiple"
          SelectionChanged="OnCollectionViewSelectionChanged">
  ...
</CollectionView>

void OnCollectionViewSelectionChanged
       (object sender, SelectionChangedEventArgs e)
{ var previous = e.PreviousSelection;
  var current = e.CurrentSelection;
  ...
}
```

С помощью значения Single свойства SelectionMode можно задать предварительный выбор элемента коллекции. Для этого для свойства SelectedItem необходимо задать значение, то есть, элемент коллекции, который должен быть выбран.

Далее приведен фрагмент программного кода, в котором задан предварительный выбор одного элемента Rocket коллекции Rockets. При этом свойство SelectedItem привязывается к свойству SelectedRocket визуальной модели представления.

```
<CollectionView ItemsSource="{Binding Rockets}"
          SelectionMode="Single"
          SelectedItem="{Binding SelectedRocket}">
  ...
</CollectionView>
```

Свойство SelectedRocket определено в классе визуальной модели представления RocketsViewModel. Если пользователь выбирает другой элемент коллекции, то значению свойства SelectedRocket будет присвоен

выбранный элемент Rocket. Свойству SelectedRocket, определенному в классе RocketsViewModel присваивается четвертый элемент коллекции Rockets, который будет предварительно выбранным при запуске программного приложения.

```
public class RocketsViewModel:
             INotifyPropertyChanged
{  ...
   public ObservableCollection<Rocket>
                  Rockets { get; private set; }
   Rocket selectedRocket;
   public Rocket SelectedRocket
   {   get { return selectedRocket;  }
       set {  if (selectedRocket != value)
            { selectedRocket = value; }  }
   }
   public RocketsViewModel()
   {   ...
      selectedRocket =
            Rockets.Skip(3).FirstOrDefault();  }
   ...
}
```

Если свойство SelectionMode имеет значение Multiple, то могут быть предварительно выбраны несколько элементов в коллекции.

Далее приведен фрагмент программного кода, в котором производится предварительный выбор нескольких элементов коллекции Rockets. При этом свойство SelectedItems привязывается к свойству SelectedRockets визуальной модели представления.

```
<CollectionView  ItemsSource="{Binding Rockets}"
          SelectionMode="Multiple"
          SelectedItems="{Binding SelectedRockets}">
   ...
</CollectionView>
```

Свойство SelectedRockets определено в классе визуальной модели представления RocketsViewModel и имеет тип ObservableCollection<object>, то есть представляет собой коллекцию элементов типа object.

Свойству SelectedRockets присваиваются второй, четвертый и пятый элементы коллекции Rockets. В элементе управления CollectionView по умолчанию выделение элементов коллекции отключено.

Однако, если в CollectionView выбор элементов включен, его можно отключить, задав для свойства SelectionMode значение None.

```
public class RocketsViewModel: INotifyPropertyChanged
{  ...
   ObservableCollection<object> selected Rockets;
   public ObservableCollection<object> SelectedRockets
   { get {return selectedRockets;}
     set { if (selectedRockets != value)
           {selectedRockets = value; }}
   }
   public RocketsViewModel()
   {  ...
     SelectedRockets =
           new ObservableCollection<object>()
     { Rockets[1], Rockets[3], Rockets[4] };
   }
   ...
}
```

Элемент управления CollectionView имеет также свойства, которые используются в том случае, когда коллекция отсутствует или в ней нет элементов:

свойство EmptyView по умолчанию имеет значение null и задает строку, ссылку или элемент управления, которые будут отображаться в случае, если свойство ItemsSource имеет значение null или если коллекция, указанная в свойстве ItemsSource имеет значение null, то есть, не имеет элементов;

свойство EmptyViewTemplate имеет по умолчанию значение null и задает шаблон, используемый для форматирования объекта, указанного в свойстве EmptyView.

Далее приведен фрагмент программного кода, в котором приведены примеры присвоения строки и элемента управления Label свойству EmptyView.

```
<CollectionView ItemsSource="{Binding EmptyRockets}"
          EmptyView="Нет данных для отображения" />

<CollectionView ItemsSource="{Binding Rockets}">
    <CollectionView.ItemTemplate>
      <DataTemplate>
        ...
      </DataTemplate>
    </CollectionView.ItemTemplate>
    <CollectionView.EmptyView>
          <Label Text="Нет данных для отображения"
              FontAttributes="Bold" TextColor="Red"/>
    </CollectionView.EmptyView>
 </CollectionView>
```

Также указан фрагмент программного кода для задания значения свойства EmptyViewTemplate. Свойству задается шаблон, который определяет внешний вид объекта, который ранее присвоен свойству EmptyView.

```
  <CollectionView ItemsSource="{Binding EmptyRockets}">
    <CollectionView.ItemTemplate>
      <DataTemplate>
        ...
      </DataTemplate>
    </CollectionView.ItemTemplate>
    <CollectionView.EmptyView>
      …Форматируемый объект…
    </CollectionView.EmptyView>
    <CollectionView.EmptyViewTemplate>
      <DataTemplate>
        <Label Text="Нет данных для отображения"
            FontSize="30" TextColor="Green"/>
      </DataTemplate>
    </CollectionView.EmptyViewTemplate>
  </CollectionView>
</Stack Layout>
```

Элемент управления CollectionView имеет два вида метода ScrollTo. Первый вид метода ScrollTo выполняет прокрутку до элемента коллекции с указанным индексом. Второй вид метода выполняет прокрутку до элемента коллекции с заданным именем. Оба метода имеют дополнительные аргументы, которые могут указывать на группу, к которой принадлежит элемент коллекции, указывать на точное расположение элемента коллекции после завершения прокрутки, а также указание на необходимость анимации прокрутки.

Элементу управления CollectionView соответствует событие ScrollToRequested, которое возникает при выполнении вызова одного методов ScrollTo.

Событию ScrollToRequested соответствует объект ScrollToRequestedEventArgs.

Кроме этого, элементу управления CollectionView соответствует событие Scrolled, которое срабатывает после завершения прокрутки.

Данному событию соответствует объект ItemsViewScrolledEventArgsОбъект.

Также элемент управления CollectionView имеет свойство ItemsUpdatingScrollMode, задающее поведение прокрутки CollectionView при добавлении новых элементов к нему.

Далее приведен фрагмент программного кода, в котором задан обработчик OnCollectionViewScrolled для события Scrolled

```
<CollectionView Scrolled="OnCollectionViewScrolled">
  ...
```

```
</CollectionView>
void OnCollectionViewScrolled
        (object sender, ItemsViewScrolledEventArgs e)
{ … }
```

Также далее рассмотрены примеры использования метода ScrollTo. Сначала выполняется прокрутка до элемента с указанным индексом. Далее рассматривается использование метода ScrollTo для прокрутки до элемента rocket с именем Tactical из коллекции Rockets. Анимация с прокруткой производится при значении аргумента animate, равного true.

```
collectionView.ScrollTo(12);
RocketsViewModel viewModel =
    BindingContext as RocketsViewModel;
Rocket rocket = viewModel.Rockets.FirstOrDefault
    (m => m.Name == "Tactical");
collectionView.ScrollTo(rocket, animate: true);
```

При прокрутке элемента в представлении точное расположение элемента после прокрутки можно указать с помощью аргумента position метода ScrollTo. Этот аргумент принимает одно из значений ScrollToPosition.

Далее приведены примеры использования метода ScrollTo с различными значениями аргумента position.

Значение ScrollToPosition.MakeVisible указывает, что элемент должен быть прокручиваться до тех пор, пока он не будет виден пользователю.

Значение ScrollToPosition.Start задает прокрутку элемента до начала области пользовательского интерфейса, в которой отображается коллекция.

Значение ScrollToPosition.Center задает прокрутку элемента до середины области пользовательского интерфейса, в которой отображается коллекция.

Значение ScrollToPosition.End задает прокрутку элемента до конца области пользовательского интерфейса, в которой отображается коллекция.

```
collectionView.ScrollTo(monkey, position: ScrollToPosition.MakeVisible);
collectionView.ScrollTo(monkey, position: ScrollToPosition.Start);
collectionView.ScrollTo(monkey, position: ScrollToPosition.Center);
collectionView.ScrollTo(monkey, position: ScrollToPosition.End);
```

Элемент управления CollectionView имеет свойство ItemsUpdatingScrollMode, которое позволяет работать с поведением прокрутки при добавлении новых элементов к коллекции. Свойству могут быть присвоены следующие значения:

значение KeepItemsInView является значением по умолчанию и задает сохранение первого элемента в коллекции при добавлении новых элементов в коллекцию;

значение KeepScrollOffset задает сохранение текущей позиции прокрутки при добавлении новых элементов в коллекцию;

значение KeepLastItemInView задает сохранения последнего элемента коллекции при добавлении новых элементов коллекции.

Элемент управления CollectionView имеет свойства HorizontalScrollBarVisibility и VerticalScrollBarVisibility. Эти свойства предназначены для задания режима отображения горизонтальной или вертикальной полосы прокрутки. Данное свойство имеет следующие значения:

значение Default является значением по умолчанию и указывает поведение полос прокрутки по умолчанию для платформы, на которой происходит работа программного приложения;

значение Always указывает на то, что полосы прокрутки будут видимы, даже если коллекция полностью помещается в той в области пользовательского интерфейса, где она отображается;

значение Never указывает на то, что полосы прокрутки будут не видны, даже если коллекция не полностью помещается в той в области пользовательского интерфейса, где она отображается.

Для управления прокруткой коллекции можно управлять конечной позицией прокрутки для того, чтобы элемент отображался полностью.

Такая функция управления прокруткой называется привязкой.

В результате элементы коллекции привязываются к позиции при завершении прокрутки. Привязка контролируется следующими свойствами класса ItemsLayout:

свойство SnapPointsType задает поведение точек привязки элементов коллекции при ее прокрутке;

свойство SnapPointsAlignment задает способ выравнивания точек привязки по элементам.

Свойство SnapPointsType может принимать следующие значения:

значение None является значением по умолчанию и указывает на то, что прокрутка не привязывается к элементам;

значение Mandatory указывает на то, что элементы привязываются к ближайшей точке привязки, а также на то, что учитывается направление инерции в случае окончания прокрутки;

значение MandatorySingle указывает на то же, что и свойство Mandatory, но применяется для прокрутки по одному элементу.

Свойство SnapPointsAlignment может иметь следующие значения:

значение Start является значением по умолчанию и указывает на то, что первый элемент коллекции будет находиться в верхней части или слева в элементе управления CollectionView.

значение Center указывает на то, что в верхней части или слева в элементе управления CollectionView отображается соответственно только нижняя или правая половина первого элемента коллекции;

значение End указывает, что последний элемент коллекции будет находиться в нижней части или справа в элементе управления CollectionView.

Прокрутка коллекций данных, состоящих из большого количества элементов, зачастую бывает неудобной, так как для прокрутки приходится использовать большие списки.

Это затрудняет навигацию по коллекции. Для того, чтобы сделать перемотку более удобной, имеется возможность группировать элементы коллекции в элементе управления CollectionView. Поддержка отображения сгруппированных данных может быть реализована с помощью следующих свойств:

свойство IsGrouped имеет значение по умолчанию false и задает необходимость отображения элементов коллекции в группах;

свойство GroupHeaderTemplate задает шаблон, используемый для формирования заголовка (верхнего колонтитула) каждой группы;

свойство GroupFooterTemplate задает шаблон, используемый для нижнего колонтитула каждой группы.

Каждая группа в коллекции является списком элементов. В каждой группе должны быть определены два фрагмента: имя группы и коллекция элементов, входящих в состав группы. Первым шагом при группировании элементов является создание класса, соответствующего одному элементу коллекции.

Далее приведен фрагмент программного кода, в котором рассматривается создание класса Rocket.

```
public class Rocket
{   public string Name { get; set; }
    public string Details { get; set; }
}
```

Затем необходимо создать класс для группы элементов и коллекцию групп.

Далее приведен фрагмент программного кода для группы RocketGroupe, имеющей в составе элементы типа Rocket. Класс RocketGroupe является дочерним от класса List<T> класса, который задает список однотипных объектов, и добавляет свойство Name, задающее имя группы. Далее создается коллекция групп с именем Rockets, в которой каждый элемент является объектом RocketGroupe.

Каждая группа типа RocketGroupe состоит из имени и коллекции элементов List<Rocket>.

```
 public class RocketGroupe: List<Rocket>
{
   public string Name { get; private set; }
   public RocketGroupe (string name, List<Rocket> R): base(R)
   {  Name = name;  }
}

public List<RocketGroupe> Rockets { get; private set; } =
new List<RocketGroupe>();
```

После этого сгруппированные данные можно добавить в коллекцию Rockets.

Далее приведен фрагмент программного кода, в котором в коллекцию Rockets добавляется группа с именем Tactical.

```
Rockets.Add(new RocketGroupe
("Tactical", new List<Rocket>
 { new Rocket
  { Name = "Rocket1", Details = "150 km" },
  new Rocket
  { Name = "Rocket2", Details = "300 km" }
 }));
```

Элемент управления CollectionView отображает сгруппированные данные с помощью присвоения свойству IsGrouped значения true.

Далее приведен фрагмент программного кода для отображения коллекции Rockets по группам

```
<CollectionView ItemsSource="{Binding Rockets}"
          IsGrouped="true">
  <CollectionView.ItemTemplate>
    <DataTemplate>
              …
    </DataTemplate>
  </CollectionView.ItemTemplate>
</CollectionView>
```

У группы можно настроить верхний колонтитул с помощью задания шаблона DataTemplate для свойства GroupHeaderTemplate элемента управления CollectionView.

Далее приведен фрагмент программного кода, в котором демонстрируется присвоение верхнему колонтитулу группы шаблона в виде элемента управления Label.

В данном элементе управления в верхнем колонтитуле отображается имя группы, а также задается форматирование элемента управления.

```
<CollectionView ItemsSource="{Binding Rockets}"
          IsGrouped="true">
  …
  <CollectionView.GroupHeaderTemplate>
    <DataTemplate>
      <Label Text="{Binding Name}"
          BackgroundColor="Red"
          FontAttributes="Bold" />
    </DataTemplate>
  </CollectionView.GroupHeaderTemplate>
</CollectionView>
```

Внешний вид нижнего колонтитула для каждой группы можно настроить, задав шаблон DataTemplate для свойства GroupFooterTemplate элемента управления CollectionView.

Далее приведен фрагмент программного кода, в котором демонстрируется присвоение заголовку в нижнем колонтитуле группы шаблона DataTemplate.

```
<CollectionView ItemsSource="{Binding Rockets}"
          IsGrouped="true">
      ...
  <CollectionView.GroupFooterTemplate>
    <DataTemplate>
     …
    </DataTemplate>
  </CollectionView.GroupFooterTemplate>
</CollectionView>
```

Для работы с элементами управления CollectionView создадим проект CollectionViewDemo для мультиплатформенного приложения Xamarin.Forms с использованием шаблона «Пустой». Создадим класс Animal. Также создадим класс AnimalsViewModel.

Откроем файл Animal.cs и наберем программный код, приведенный далее. В программном коде приведены свойства Name, Location, ImageUrl класса Animal, на которые будет производиться ссылаться из файлов MainPage.xaml и AnimalsViewModel.cs.

```
using System;
using System.Collections.Generic;
using System.Text;

namespace CollectionViewDemo
{
  public class Animal
  {
    public string Name { get; set; }
    public string Location { get; set; }
    public string ImageUrl { get; set; }
  }
}
```

Откроем файл AnimalsViewModel.cs и наберем программный код, приведенный далее.

В программном коде сначала объявляются свойство source, доступное только для чтения. Свойству будет присваиваться список, состоящий из элементов типа Animal.

В методе AnimalsViewModel производится присвоение списка из элементов типа Animal переменной source.

Производится создание визуальной коллекции элементов Monkeys, которая состоит из 9 элементов типа Monkey.

Отметим, что в программном коде значения свойства ImageUrl типа String для удобства демонстрации приведены в виде многострочного текста. При наборе программного кода в среде разработки необходимо набирать текст в одну строку. В противном случае картинки, загружаемые в элементы коллекции Monkeys с использованием свойств ImageUrl, не будут отображаться.

```
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Windows.Input;
using Xamarin.Forms;

namespace CollectionViewDemo
{
  public class AnimalsViewModel
  {
    readonly IList<Animal> source;
    public ObservableCollection<Animal> Animals { get; private set; }

    public AnimalsViewModel()
    {
      source = new List<Animal>();
      CreateAnimalCollection();
    }
    void CreateAnimalCollection()
    {
      source.Add(new Animal
      {
        Name = "Капучин",
        Location = "Центральная и Южная Америка",
        ImageUrl = "https://upload.wikimedia.org/wikipedia/" +
        "commons/thumb/4/40/Capuchin_Costa_Rica.jpg/" +
        "200px-Capuchin_Costa_Rica.jpg"
      });

      source.Add(new Animal
      {
        Name = "Голубая обезьяна",
        Location = "Центральная и восточная Африка",
        ImageUrl = "https://upload.wikimedia.org/wikipedia/" +
        "commons/thumb/8/83/BlueMonkey.jpg/220px-BlueMonkey.jpg"
      });
```

```
source.Add(new Animal
{
   Name = "Золотистый львиный тамарин",
   Location = "Бразилия",
   ImageUrl = "https://upload.wikimedia.org/wikipedia/" +
   "commons/thumb/8/87/Golden_lion_tamarin_portrait3.jpg/" +
   "220px-Golden_lion_tamarin_portrait3.jpg"
});

source.Add(new Animal
{
   Name = "Ревуны",
   Location = "Центральная и Южная Америка",
   ImageUrl = "https://upload.wikimedia.org/wikipedia/" +
   "commons/thumb/0/0d/Alouatta_guariba.jpg/" +
   "200px-Alouatta_guariba.jpg"
});

source.Add(new Animal
{
   Name = "Японский макак",
   Location = "Япония",
   ImageUrl = "https://upload.wikimedia.org/wikipedia/" +
   "commons/thumb/c/c1/Macaca_fuscata_fuscata1.jpg/" +
   "220px-Macaca_fuscata_fuscata1.jpg"
});

source.Add(new Animal
{
   Name = "Носач",
   Location = "Борнео",
   ImageUrl = "https://upload.wikimedia.org/wikipedia/" +
   "commons/thumb/e/e5/Proboscis_Monkey_in_Borneo.jpg/" +
   "250px-Proboscis_Monkey_in_Borneo.jpg"
});

source.Add(new Animal
{
   Name = "Красноголовый доук",
   Location = "Вьетнам и Лаос",
   ImageUrl = "https://upload.wikimedia.org/wikipedia/" +
   "commons/thumb/9/9f/Portrait_of_a_Douc.jpg/" +
   "159px-Portrait_of_a_Douc.jpg"
});

source.Add(new Animal
{
   Name = "Черная курносая обезьяна",
   Location = "Китай",
```

```
            ImageUrl = "https://upload.wikimedia.org/wikipedia/" +
            "commons/thumb/5/59/RhinopitecusBieti.jpg/" +
            "320px-RhinopitecusBieti.jpg"
        });

        source.Add(new Animal
        {
            Name = "Гелада",
            Location = "Эфиопия",
            ImageUrl = "https://upload.wikimedia.org/wikipedia/" +
            "commons/thumb/1/13/Gelada-Pavian.jpg/" +
            "320px-Gelada-Pavian.jpg"
        });
        Animals = new ObservableCollection<Animal>(source);
    }
  }
}
```

Откроем файл MainPage.xaml и наберем программный код, приведенный далее. В программном коде определено, что для страницы типа ContentPage используется визуальная коллекция данных AnimalsViewModel, которая формируется с помощью программного кода, приведенного в файле AnimalsViewMode.cs.

Внутри страницы расположен макет типа StackLayout, содержащий макет Grid и элемент управления CollectionView.

Макет Grid предназначен для отображения названий элементов коллекции, загруженной в элемент управления CollectionView. В макете Grid содержится таблица из двух строк и двух столбцов. В первой строке размещены два элемента управления Label. С помощью этих элементов управления отображается название элемента коллекции, выбранного в предыдущий момент времени. При этом для выдачи имени элемента, выбранного в предыдущий момент времени, используется элемент управления Label с именем previousSelectedItemLabel. Во второй строке также расположены два элемента управления Label. С помощью них отображается название элемента коллекции, выбранного в текущий момент времени. При этом для выдачи имени элемента, выбранного в текущий момент времени, используется элемент управления Label с именем currentSelectedItemLabel.

В элемент управления CollectionView с помощью привязки Binding загружается визуальная коллекция данных Animals, состоящая из 9 элементов. Для элемента управления CollectionView с помощью свойства ItemsLayout задается макет в виде вертикального списка. При этом, с помощью свойства ItemSpacing задается расстояние между элементами отображаемой коллекции. С помощью свойства SelectionMode задается режим выбора только одного элемента в коллекции. В программном коде задается ссылка на обработчика события SelectionChanged, которое происходит в случае смены выбираемого элемента коллекции.

Программный код обработчика OnCollectionViewSelectionChanged приведен в файле MainPage.xaml.cs.

Внешний вид каждого элемента коллекции в элементе управления CollectionView задается путем присвоения свойству ItemTemplate шаблона данных DataTemplate. Шаблон данных DataTemplate содержит макет Grid с таблицей из двух строчек и столбцов. В первом столбце таблицы объединяются две строки, и там размещается элемент управления Image, в который с помощью свойства ImageUrl загружается картинка. Свойству ImageUrl каждого элемента коллекции в соответствии с файлом AnimalsViewMode.cs соответствует ссылка на источник в Интернете.

В первой строчке второго столбца таблицы в макете Grid отображается элемент управления Label для вывода текста, соответствующего свойству Name элемента коллекции. Во второй строке второго столбца отображается элемент управления Label для вывода текста, соответствующего свойству Location элемента коллекции. При этом производится привязка свойства Text элементов управления Label к свойствам Name и Location элемента коллекции Animals.

```
<ContentPage xmlns="http://xamarin.com/schemas/2014/forms"
        xmlns:x="http://schemas.microsoft.com/winfx/2009/xaml"
        x:Class="CollectionViewDemo.MainPage"
        BackgroundColor="Bisque">
  <StackLayout Margin="20">
    <Grid>
      <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
        <ColumnDefinition Width="Auto" />
        <ColumnDefinition Width="Auto" />
      </Grid.ColumnDefinitions>
      <Label Text="Предыдущий выбранный элемент:"
          TextColor="Black"
          FontAttributes="Bold"/>
      <Label x:Name="previousSelectedItemLabel"
          Grid.Column="1"
          TextColor="Red"
          FontAttributes="Bold"/>
      <Label Grid.Row="1"
          Text="Текущий выбранный элемент:"
         TextColor="Black"
          FontAttributes="Bold"/>
      <Label x:Name="currentSelectedItemLabel"
          Grid.Row="1"
          Grid.Column="1"
          TextColor="Red"
          FontAttributes="Bold"/>
    </Grid>
```

```
    <CollectionView ItemsSource="{Binding Animals}"
                    SelectionMode="Single"
                    SelectionChanged=
                    "OnCollectionViewSelectionChanged">
      <CollectionView.ItemsLayout>
        <LinearItemsLayout Orientation="Vertical"
                           ItemSpacing="10"/>
      </CollectionView.ItemsLayout>
      <CollectionView.ItemTemplate>
        <DataTemplate>
          <Grid Padding="10"
                BackgroundColor="Aqua">
            <Grid.RowDefinitions>
              <RowDefinition Height="Auto" />
              <RowDefinition Height="Auto" />
            </Grid.RowDefinitions>
            <Grid.ColumnDefinitions>
              <ColumnDefinition Width="Auto" />
              <ColumnDefinition Width="Auto" />
            </Grid.ColumnDefinitions>
            <Image Grid.RowSpan="2"
                   Source="{Binding ImageUrl}"
                   Aspect="AspectFill"
                   HeightRequest="60"
                   WidthRequest="60" />
            <Label Grid.Column="1"
                   Text="{Binding Name}"
                   FontAttributes="Bold"
                   TextColor="Black"
                   FontSize="18"/>
            <Label Grid.Row="1"
                   Grid.Column="1"
                   Text="{Binding Location}"
                   FontAttributes="Italic"
                   VerticalOptions="End"
                   TextColor="Black"
                   FontSize="18"/>
          </Grid>
        </DataTemplate>
      </CollectionView.ItemTemplate>
    </CollectionView>
  </StackLayout>
</ContentPage>
```

Откроем файл MainPage.xaml.cs и наберем программный код, приведенный далее.

В программном коде сначала производится привязка к визуальной модели данных, приведенной в файле AnimalsViewModel.cs. Также происходит объявление метода UpdateSelectionData. Обработчик

OnCollectionViewSelectionChanged, к которому происходит обращение из файла MainPage.xaml, принимает в качестве аргументов элемент управления CollectionView, а также элементы коллекции e.PreviousSelection и e.CurrentSelection, выбранные в предыдущий и текущий моменты времени.

Элементы коллекции передаются далее для обработки в метод UpdateSelectionData. В данном методе сначала производится извлечение имен переданных элементов коллекции и их присвоение переменным строкового типа previous и current. Затем производится присвоение значений переменных previous и current свойствам Text элементов управления Label с именами previousSelectedItemLabel и currentSelectedItemLabel. Элементы управления Label с указанными именами сформированы в программном коде в файле MainPage.xaml. При этом, если в процессе работы программного приложения выбор элементов коллекции пока еще не состоялся, то в элементах управления Label с именами previousSelectedItemLabel и currentSelectedItemLabel отображается текст «Не выбран».

```
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.ComponentModel;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Windows.Input;
using Xamarin.Forms;

namespace CollectionViewDemo
{
   public partial class MainPage : ContentPage
   {
      public MainPage()
      {
         InitializeComponent();
         BindingContext = new AnimalsViewModel();
         UpdateSelectionData(Enumerable.Empty<Animal>(),
                        Enumerable.Empty<Animal>());
      }
      void OnCollectionViewSelectionChanged(object sender,
                        SelectionChangedEventArgs e)
      {
         UpdateSelectionData(e.PreviousSelection,
                        e.CurrentSelection);
      }
      void UpdateSelectionData(IEnumerable<object> previousSelectedItems,
                        IEnumerable<object> currentSelectedItems)
      {
         string previous =
            (previousSelectedItems.FirstOrDefault() as Animal)?.Name;
         string current =
            (currentSelectedItems.FirstOrDefault() as Animal)?.Name;
```

```
            previousSelectedItemLabel.Text =
               string.IsNullOrWhiteSpace(previous) ? "Не выбран" : previous;
            currentSelectedItemLabel.Text =
               string.IsNullOrWhiteSpace(current) ? "Не выбран" : current;
        }
    }
}
```

Запустим программное приложение для работы с элементом управления CollectionView. При запуске программного приложения в верхней части пользовательского интерфейса отображается информация об элементах коллекции, выбранных в предыдущий и текущий моменты времени. В начале работы программного приложения выбор элементов коллекции пока еще не производился. Поэтому в пользовательском интерфейсе отображаются сообщения «Не выбран».

В пользовательском интерфейсе отображается коллекция в виде вертикального списка, состоящая из 9 элементов. Каждый элемент коллекции имеет одинаковый внешний вид, обусловленный рассмотренным ранее шаблоном. Список может прокручиваться вверх и вниз.

Предусматривается возможность выбора одного элемента коллекции. При выборе элемента коллекции в верхней части пользовательского интерфейса происходит изменение информации об элементах коллекции, выбранных в предыдущий и текущий моменты времени.