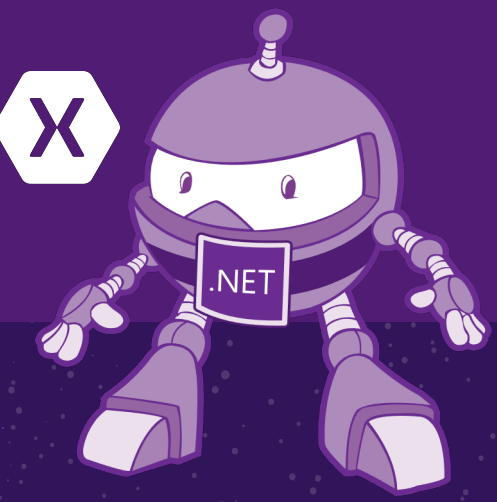
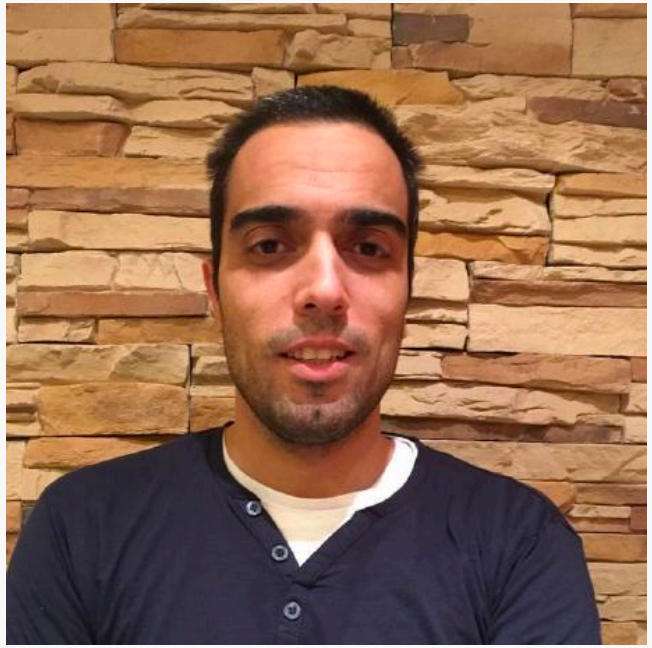

.NET Conf
"Focus on Xamarin"
.NET
focus.dotnetconf.net

Visualize Your Data (CollectionView, CarouselView, & Beyond)

Gerald Versluis
Javier Suárez



Who are you listening to?



Javier Suárez

Software Engineer
Xamarin.Forms Team

@jsuarezruiz

Gerald Versluis

Software Engineer
Xamarin.Forms Team

@jfversluis

The agenda

CollectionView

- Layout options
- Snap Points
- ScrollTo
- EmptyView
- Selection
- Grouping
- Item Spacing
- Scroll mode
- Header/Footer
- Nested

CarouselView

- Layout options
- Snap Points
- Scroll Options
- Peek and Spacing
- EmptyView

IndicatorView

- Use with CarouselView
- Customization

RefreshView

- Pull To Refresh
- Customization

SwipeView

- Directions
- Modes
- BehaviorOnInvoked
- Command
- Use in Collections
- Custom SwipeItem

CollectionView

.NET

CollectionView

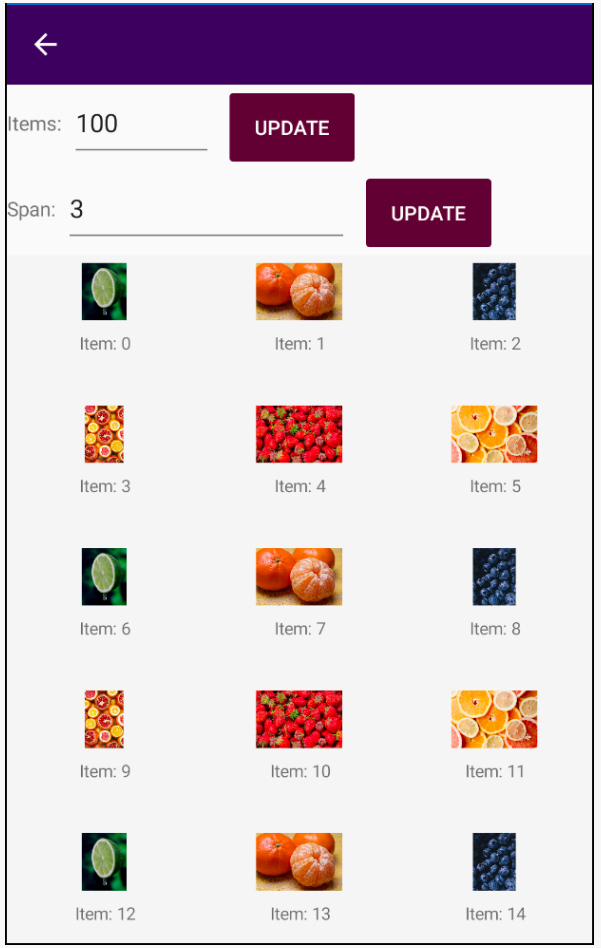
`CollectionView` is a view for presenting lists of data using different layout specifications. It aims to provide a more flexible, and performant alternative to **`ListView`**.

```
<CollectionView ItemsSource="{Binding Monkeys}">
    <CollectionView.ItemsLayout>
        <LinearItemsLayout Orientation="Vertical" />
    </CollectionView.ItemsLayout>
    <CollectionView.ItemTemplate>
        <DataTemplate>
            ...
        </DataTemplate>
    </CollectionView.ItemTemplate>
</CollectionView>
```

CollectionView Customizations

Out of the box this is what you can do with **`CollectionView`**:

- Data binding with templates
- Different layouts
  - Horizontal, Vertical and Grid
- Item selection
- Empty view
- Scrolling
  - Event with lots of info
  - **`ScrollTo`** method
- And much more!



DEMO:

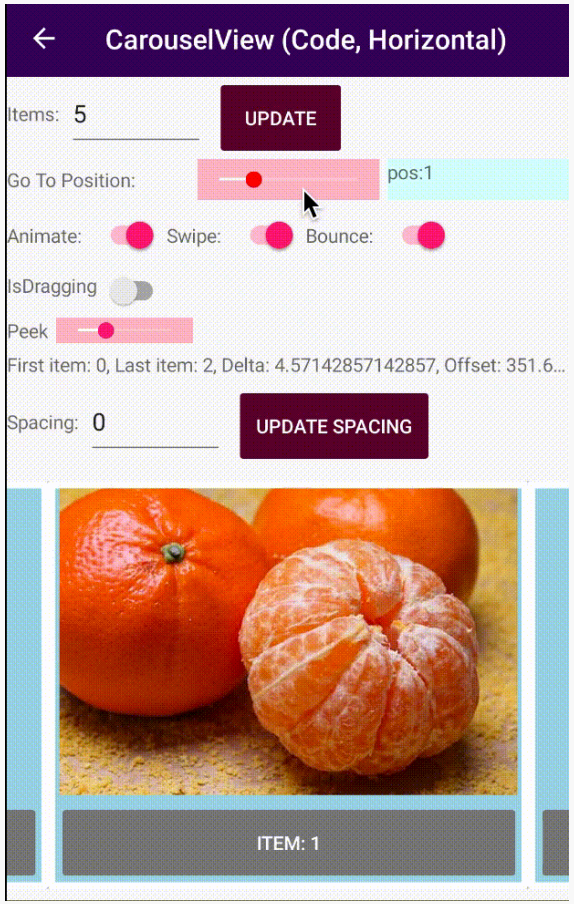
CollectionView

CarouselView

.NET

CarouselView (preview)

`CarouselView` is a view for presenting data in a scrollable layout, where users can swipe to move through a collection of items.



CarouselView (preview)

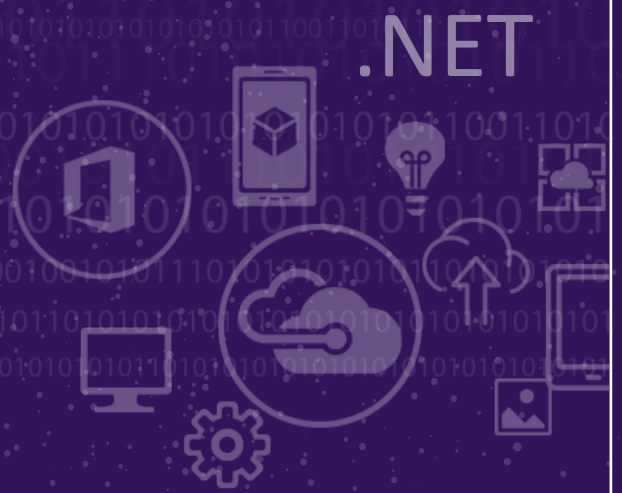
`CarouselView` is built on top of `CollectionView`, so you get a lot of the functionality for free.

- Show data in a visually appealing way, very popular in modern apps.
- More suitable for limited length data.
- Also use orientation, layout, data templates, etc.
- Works perfectly together with `IndicatorView.`

Note: in preview right now, needs the experimental flag.

DEMO:

CarouselView



IndicatorView

IndicatorView

The **`IndicatorView`** allows to indicate which is the current element of a collection.

Normally, the **`IndicatorView`** is associated with a **`CarouselView`**. We associate the controls using the **`IndicatorView`** property from the **`CarouselView`**.

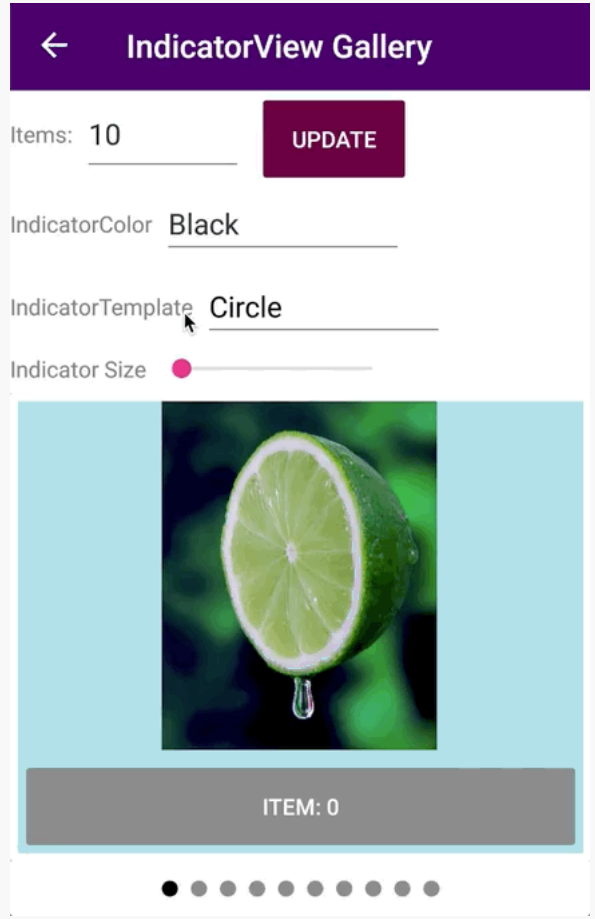
```
var indicatorView = new IndicatorView
{
    HorizontalOptions = LayoutOptions.Center,
    IndicatorColor = Color.Gray,
    SelectedIndicatorColor = Color.Black,
    IndicatorsShape = IndicatorShape.Square
};

carouselView.IndicatorView = indicatorView;
```

Note: in preview right now, needs the experimental flag.

IndicatorView Possibilities

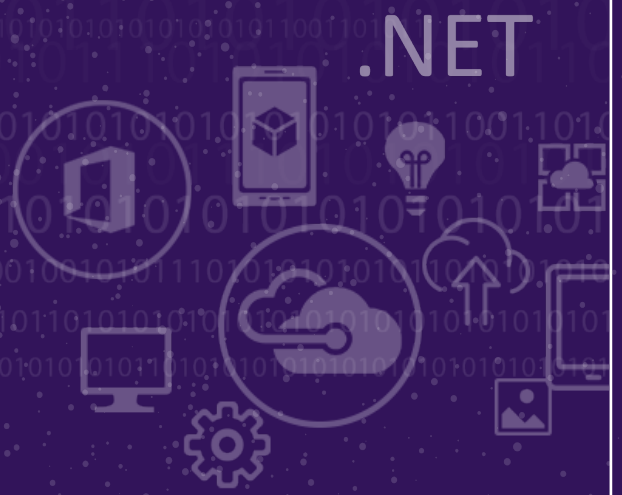
`IndicatorView` has a variety of customization options:

- **IndicatorColor**: Is the color of the indicators.
- **SelectedIndicatorColor**: the color of the indicator that represents the current item in the CarouselView.
- **IndicatorShape**: the shape of each indicator. By default can use circle or rectangle.
- **IndicatorTemplate**: the template that defines the appearance of each indicator.
- **IndicatorSize**: the size of the indicators.



DEMO:

IndicatorView



RefreshView

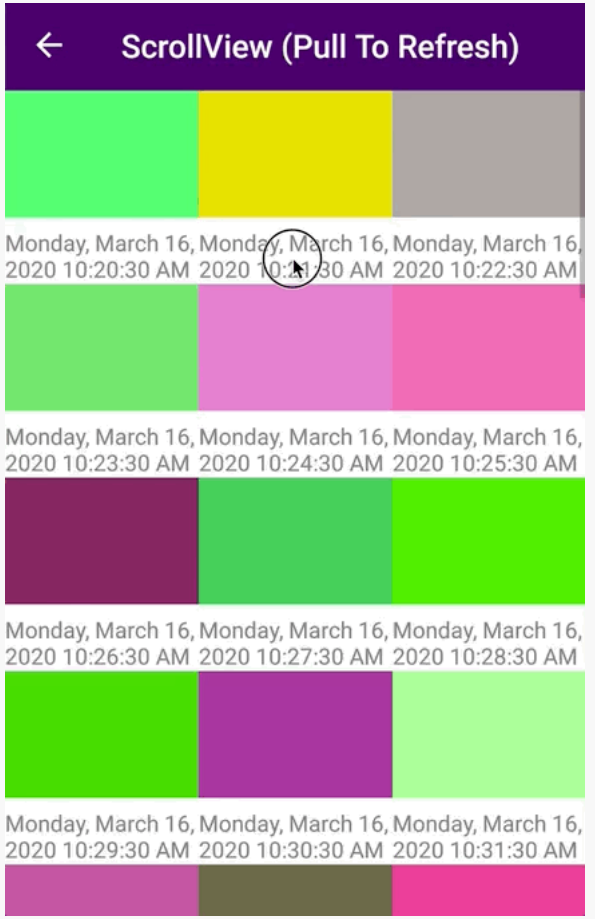
Pull To Refresh

Pull-to-refresh lets a user pull down on a scrollable content in order to retrieve more data. The main control to allow pull to refresh is the **`RefreshView`**, which you place as a wrapper around scrollable content that the user pulls to trigger a refresh.

Implement Pull To Refresh

To add pull-to-refresh functionality to a list requires just two steps:

1. Wrap your scrollable content (list) in a `RefreshView` control.
2. Handle the **`Command`** to refresh your content.

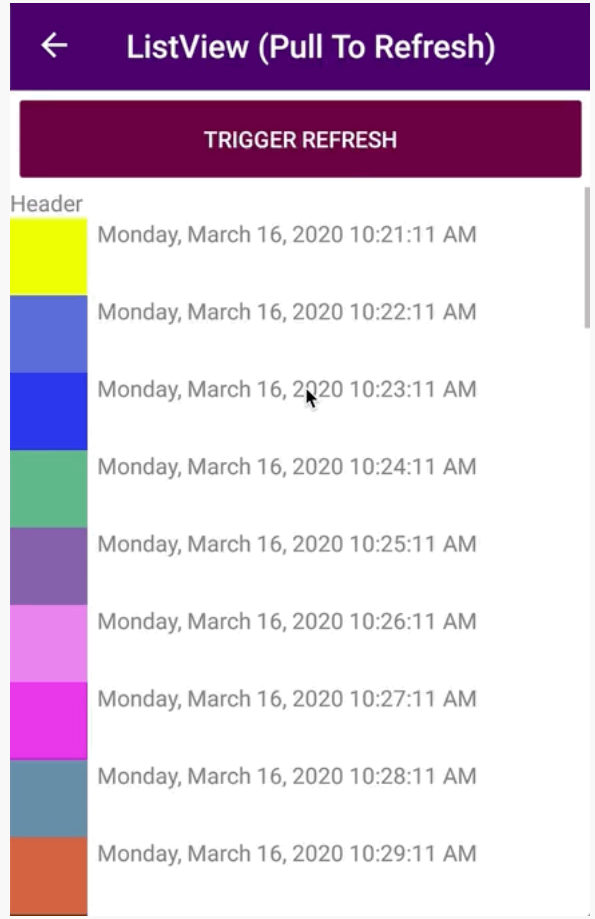



```
<RefreshView
    IsRefreshing="{Binding IsRefreshing}"
    Command="{Binding RefreshCommand}">
    <ListView
        ItemsSource="{Binding Items}">
        …
    </ListView>
</RefreshView>
```

Customize the Refresh Indicator

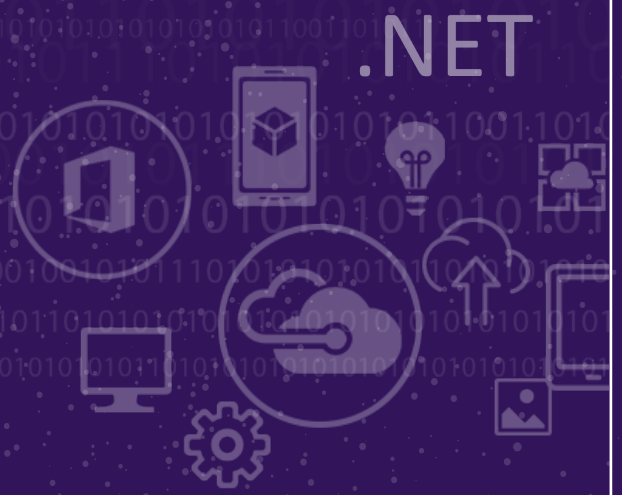
To customize the background color of the refresh indicator as well as the color of the indicator can use the **BackgroundColor** and **RefreshColor** properties respectively.

```
<RefreshView
    IsRefreshing="{Binding IsRefreshing}"
    BackgroundColor="Red"
    RefreshColor="Yellow"
    Command="{Binding RefreshCommand}"
    HorizontalOptions="FillAndExpand"
    VerticalOptions="FillAndExpand" />
```



DEMO:

RefreshView



SwipeView

SwipeView

Swipe commanding is an accelerator for context menus that lets users easily access common menu actions, without needing to change states within the app.

How does Swipe work?

`SwipeView` commanding has two modes: **Reveal** and **Execute**. It also supports four different swipe directions: **up**, **down**, **left**, and **right**.

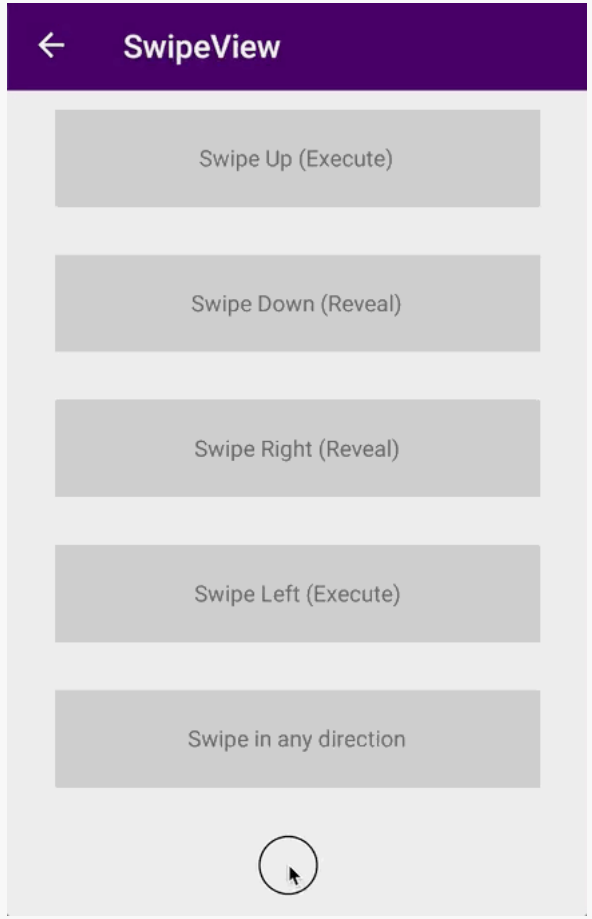
- In **Reveal mode**, the user swipes an item to open a menu of one or more commands and must explicitly tap a command to execute it.
- In **Execute mode**, the user swipes an item open to reveal and execute a single command with that one swipe. If the user releases the item being swiped before they swipe past a threshold, the menu closes and the command is not executed. If the user swipes past the threshold and then releases the item, the command is executed.

Note: in preview right now, needs the experimental flag.

Swipe Directions

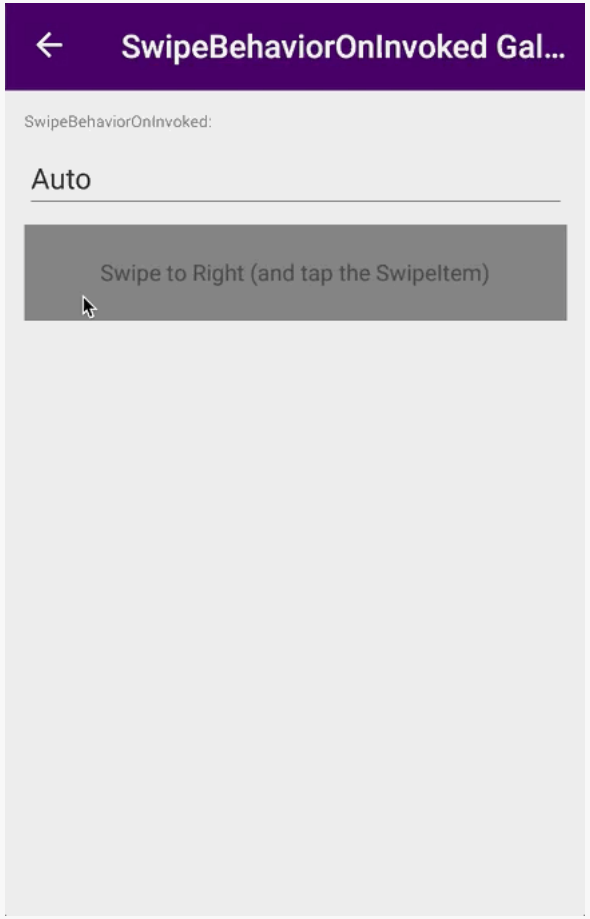
Swipe works in all cardinal directions: **up**, **down**, **left**, and **right**. Each swipe direction can hold its own swipe items or content, but only one instance of a direction can be set at a time on a single swipe-able element.

```
<SwipeView>
    <SwipeView.LeftItems>
        <SwipeItems Mode="Reveal">
            <SwipeItem
                Text="Delete"
                Icon="delete.png"
                BackgroundColor="Red"
                Invoked="OnInvoked"/>
        </SwipeItems>
    </SwipeView.LeftItems>
    <SwipeView.Content>
        <Grid BackgroundColor="LightGray">
            <Label
                HorizontalOptions="Center"
                VerticalOptions="Center"
                Text="Swipe Right (Reveal)"/>
        </Grid>
    </SwipeView.Content>
</SwipeView>
```



Swipe BehaviorOnInvoked

In situations where you simply want to perform an action and then maintain the swipe open, you can set the **`BehaviorOnInvoked`** property one of the **`SwipeBehaviorOnInvoked`** enum values.



- **Auto**: The opened swipe item will collapse when invoked.
- **Close**: When the item is invoked, the SwipeView will always collapse and return to normal, regardless of the mode.
- **RemainOpen**: When the item is invoked, the SwipeView will always stay open.

Handle an invoked swipe command

To act on a swipe command, you handle its **`Invoked`** event.

```
<SwipeItem
    Text="Delete"
    Icon="delete.png"
    BackgroundColor="Red”
    Command="{StaticResource DeelteCommand}"
    Invoked="OnDeleteInvoked"/>

void OnDeleteInvoked(object sender, EventArgs e)
{

}
```

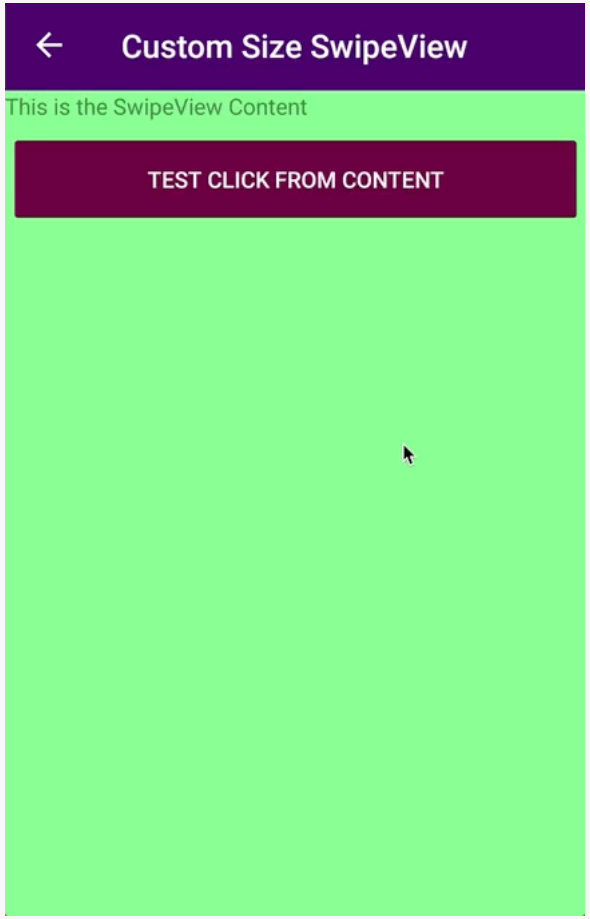
SwipeView in Collections

SwipeView in combination with **BindableLayout**, **CollectionView**, **CarouselView**, **ListView** or any control managing a collection allow to manage **contextual options**.


```
<CollectionView
    ItemsSource="{Binding Items}">
    <CollectionView.ItemTemplate>
        <DataTemplate>
            <SwipeView>
            ...
            </SwipeView>
        </DataTemplate>
    </CollectionView.ItemTemplate>
</CollectionView>
```

Custom SwipeItems

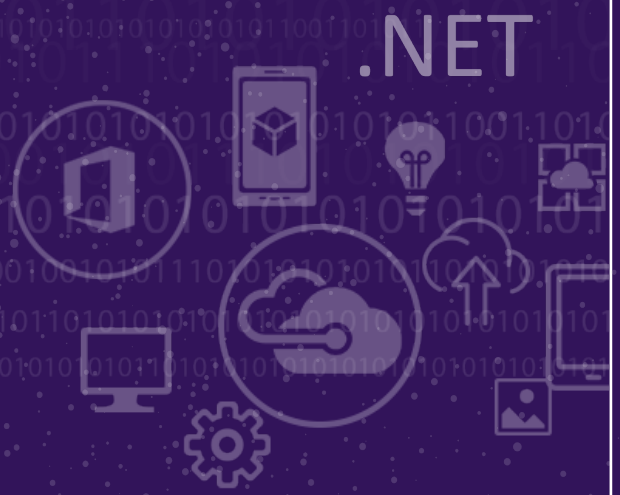
In addition to `SwipeItem` we have **SwipeItemView** that allows us to include a `View` as content, that is, we can create fully customized `SwipeItems`.




```
<SwipeView.TopItems>
    <SwipeItemView>
        <StackLayout
            BackgroundColor="LightSkyBlue"
            HeightRequest="100">
            <Label
                Text="This is the TopItems Content"
                Margin="12"/>
            <Button
                Text="Click me!"
                Clicked="OnButtonClicked"/>
        </StackLayout>
    </SwipeItemView>
</SwipeView.TopItems>
```

DEMO:

SwipeView



Upcoming features

Do you want to know what functionality will come soon to this group of controls?

- CarouselView
  - Cyclic CarouselView.
  - CarouselView VisualStates.
- SwipeView
  - Modify the swipe threshold.
  - Programmatically open and close the SwipeView.
  - Hide or disable SwipeItems.

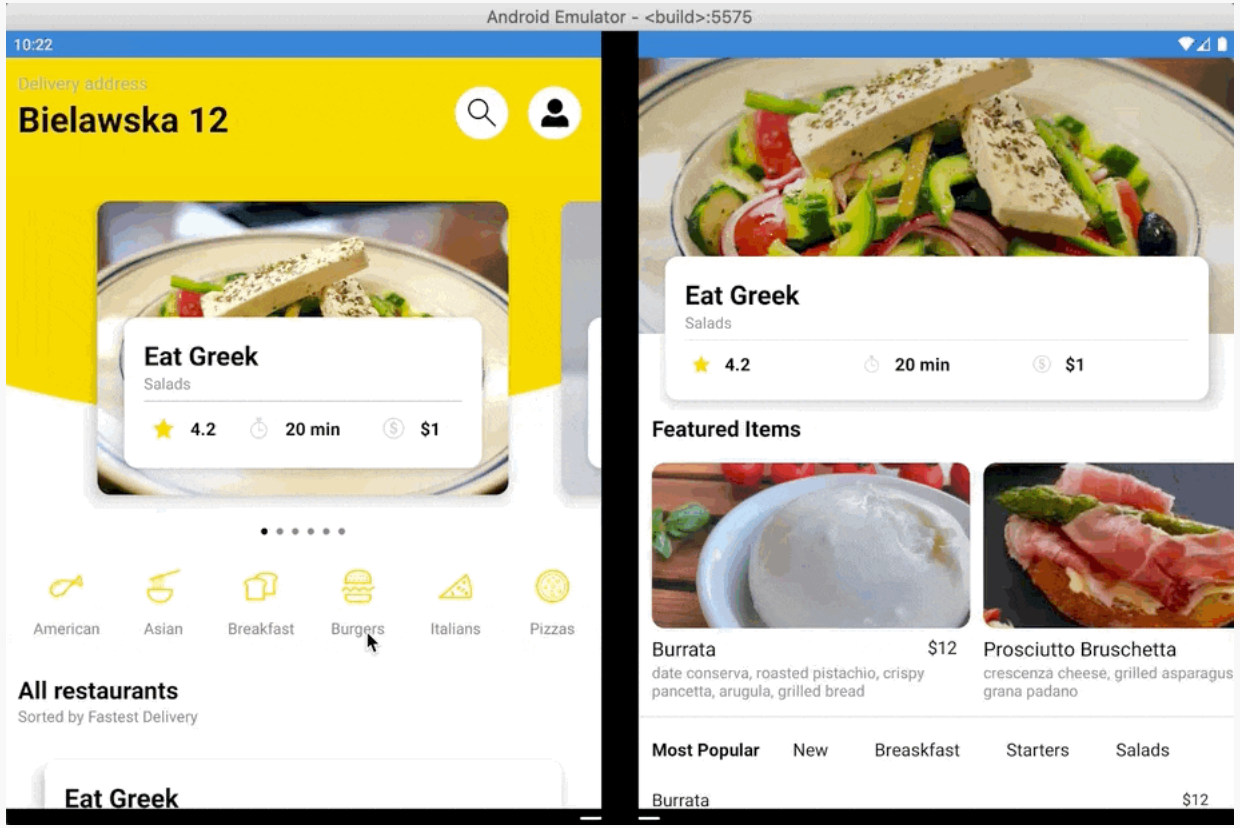
```
<VisualStateManager.VisualStateGroups>
    <VisualStateGroup x:Name="CarouselViewStates">
        <VisualState x:Name="CurrentItem">
            <VisualState.Setters>
                <Setter Property="Opacity" Value="1" />
            </VisualState.Setters>
        </VisualState>
        <VisualState x:Name="PreviousItem">
            <VisualState.Setters>
                <Setter Property="Opacity" Value="0.7" />
            </VisualState.Setters>
        </VisualState>
        <VisualState x:Name="NextItem">
            <VisualState.Setters>
                <Setter Property="Opacity" Value="0.7" />
            </VisualState.Setters>
        </VisualState>
        <VisualState x:Name="DefaultItem">
            <VisualState.Setters>
                <Setter Property="Opacity" Value="0.2" />
            </VisualState.Setters>
        </VisualState>
    </VisualStateGroup>
</VisualStateManager.VisualStateGroups>
```

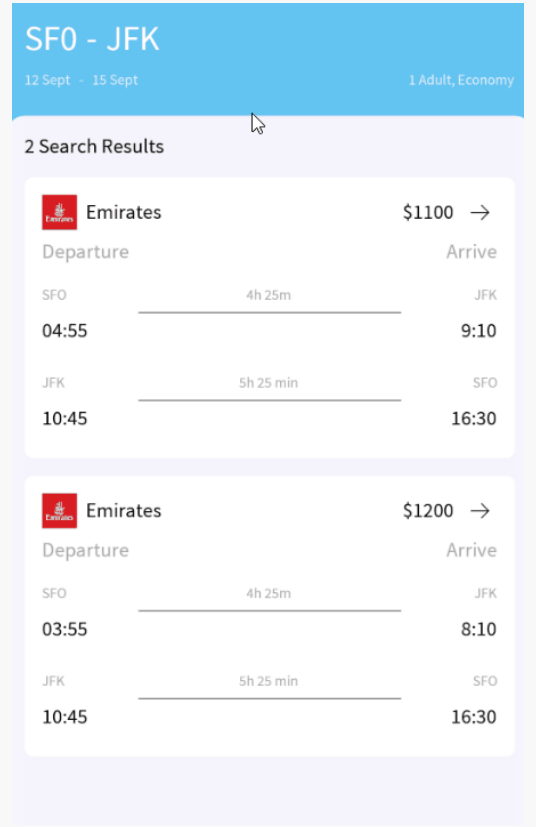
Good Looking UI samples

.NET

Good Looking UI Samples

Now that we have taken a look at the `CollectionView`, `CarouselView`, `RefreshView` and `SwipeView` concepts, let's see how to use this controls in some Good Looking UI samples.



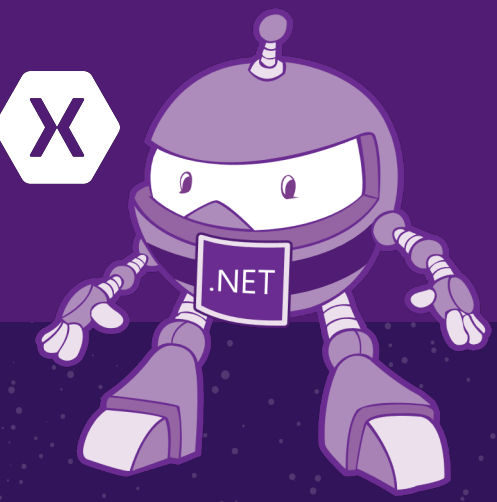


DEMO:

GoodLooking UI Samples

Visualize Your Data (CollectionView, CarouselView, & Beyond)

https://github.com/jsuarezruiz/FocusOnXamarin
https://github.com/jsuarezruiz/xamarin-forms-goodlooking-UI




.NET Conf
"Focus on Xamarin"
.NET
focus.dotnetconf.net