Xamarin.Forms

Adding some polish, performance and stability to your Forms application

Background

Rob DeRosa

Customer Success Engineer, Enterprise

Xamarin Sport

Xamarin Sport

Leaderboard app for managing rankings within leagues

- iOS and Android apps
- XAML w/ Bindings, Converters, Styles
- Custom Renderers, Plugins and Animations
- Messaging Center, Dependency Service
- XTC UITest w/ single code base, Insights
- Powered by Azure Mobile Service C# backend

Application Design Goals

- Simplified registration/authentication
- Athlete's can join multiple leagues
- View current rank within ladder
- Athlete stats & challenge history
- Facilitate challenge flow (accept, nudge, post results)

Xamarin Sport

435 lines	413 lines	12,471 lines
3.3%	3.1%	**93.6%**

http://github.com/xamarin/sport

Animations

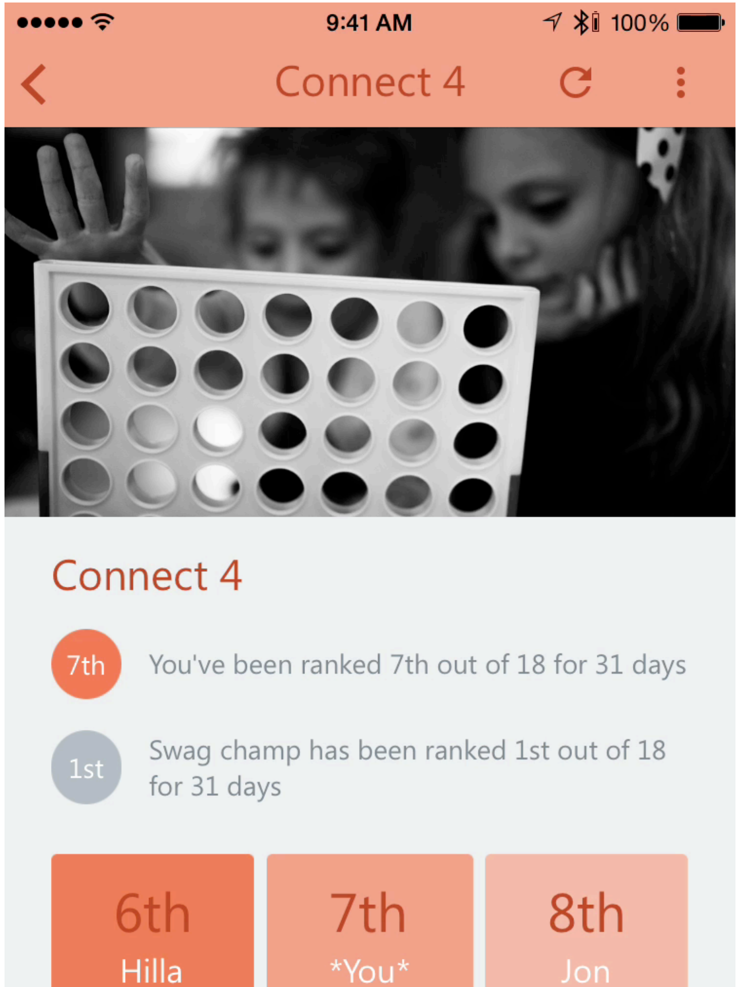
- Super easy to use
- 4 basic animations are supported
  - Scale – controls size
  - Translate – controls position
  - Fade – controls alpha/opacity
  - Rotate – controls rotation
- Supports easing (Bounce, Cubic, Linear, Sin)
- Combine animations by not using **`await`**
- Uses the underlying OS animation libraries

Animations

```
var speed = (uint)App.AnimationSpeed;
var ease = Easing.SinIn;

//Fade out and move the buttons off the right side of the screen
var pushRect = new Rectangle(Content.Width, btnPush.Bounds.Y, btnPush.Bounds.Width, btnPush.Height);
btnPush.FadeTo(0, speed, ease);
await btnPush.LayoutTo(pushRect, speed, ease);

var contRect = new Rectangle(Content.Width, btnCont.Bounds.Y, btnCont.Bounds.Width, btnCont.Height);
btnCont.FadeTo(0, speed, ease);
await btnCont.LayoutTo(contRect, speed, ease);
```

Parallax Effect

```
void Parallax()
{
    if(_imageHeight <= 0)
        _imageHeight = photoImage.Height;

    var y = scrollView.ScrollY * .4;
    if(y >= 0)
    {
        //Move the Image's Y coordinate a fraction
        //of the ScrollView's Y position
        photoImage.Scale = 1;
        photoImage.TranslationY = y;
    }
    else
    {
        //Calculate a scale that equalizes the height vs scroll
        double newHeight = _imageHeight + (scrollView.ScrollY * -1);
        photoImage.Scale = newHeight / _imageHeight;
        photoImage.TranslationY = scrollView.ScrollY / 2;
    }
}
```



Connect 4

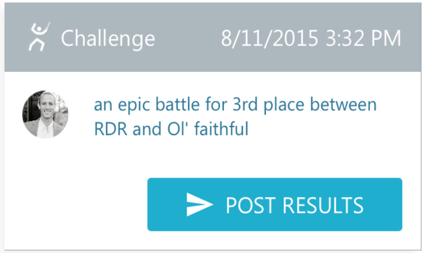
7th You've been ranked 7th out of 18 for 31 days

1st Swag champ has been ranked 1st out of 18 for 31 days

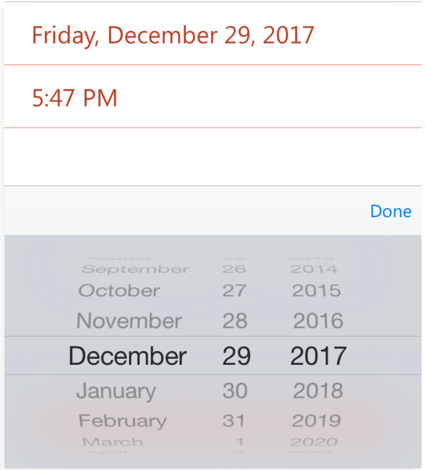
Custom, Reusable Controls

- Equivalent to a WPF/ASP.NET User Control
- Use Bindable Properties to allow consumers to bind values
- Templated controls can allow for inner content
- Subclass `ContentView`

- Use layers for precise layout - position and size
- Use opacity on controls to handle a user event
- Try to keep layers to a reasonable amount


Challenge 8/11/2015 3:32 PM

an epic battle for 3rd place between RDR and Ol' faithful


Friday, December 29, 2017

5:47 PM

Absolute Layout

- Used to control size and position of child elements
- `LayoutBounds` is based on `LayoutFlags` value
  - `SizeProportional`, `PositionProportional`, `All`
- Element's `LayoutBounds` are proportional to the AbsoluteLayout's bounds
- The `x` and `y` anchor points are interpolated as the child elements position changes

Code

Performance: ListView

- Use Headers and Footers to add additional UI elements before and after the contents of a list – do not add a `ListView` to a `ScrollView`
- Use `AbsoluteLayouts` in cell templates when you can – they are fast
- Eager-load and apply binding context to pages with `ListViews` on a background thread
  - This is true for any long-loading view
  - Do not apply binding context after adding to the visual tree
- Use the `RecycleElement` caching strategy when applicable

Performance: ListView Caching Strategy

`RetainElement`

- Default (for now)
- Creates 1 Forms cell instance for each row in the bound dataset

`RecycleElement`

- Non-default
- Creates 1 Forms cell instance for each visual element

`RecycleElement` is your best bet in most cases

Performance: ListView Caching Strategy

`RetainElement` can have better performance in the following scenarios

- Lists with fewer cells – ~300 records or less
- Cell templates that contain more than ~25 bindings
- When swapping out root elements based on the binding context
  - This may be common when trying to show and hide root elements based on a condition

Performance: ListView Caching Strategy

There is no `CachingStrategy` property on `ListView`.

Because the caching strategy can only be set once, it is passed in as a constructor argument and cannot be modified.

The XAML compiler does some magic to recognize the markup value and initialize the instance with it.

There are 3 different ways you may need to specify `CachingStrategy`.

Performance: ListView Caching Strategy

Typical `ListView` instance in XAML

```
<ListView  CachingStrategy="RecycleElement"   ...>
    ...
</ListView>
```

Subclassing `ListView` in C# (cannot specify in XAML)

```
public partial class CustomListView  : ListView
{
    public CustomListView(ListViewCachingStrategy  strategy) : base(strategy)
    {
        InitializeComponent();
    }
}
```

Instance of a subclassed `ListView` overriding default

```
<my:CustomListView>
    <x:Arguments>
        <ListViewCachingStrategy>RecycleElement</ListViewCachingStrategy>
    </x:Arguments>
    ...
</my:CustomListView>
```

Performance: ListView Caching Strategy

There is no `CachingStrategy` property on `ListView`.

Because the caching strategy can only be set once, it is passed in as a constructor argument and cannot be modified.

The XAML compiler does some magic to recognize the markup value and initialize the instance with it.

There are 3 different ways to specify `CachingStrategy`.

XAMLC

XAMLC is a compiler for XAML markup and has a few benefits:

- Catches XAML errors at compile-time instead of runtime
- Reduction in resources because the .xaml files no longer need to be included
- Inflation of views is increased 10x
  - no impact on reflection ☹
  - big impact when used with `RetainElement`
- Must be manually enabled but will become the default option in about a year

Rob’s Tips and Tricks

By creating non-running tasks, we can run them through a proxy method to:

- Check conditions (is the device connected to the internet?)
- Catch and report all exceptions to a central error handler
- Manage cancellation and tokens

Use a `Busy` object to avoid forgetting to set `IsBusy` back to false

Called from within a ViewModel that has a `IsBusy(bool)` property

```
async public Task RefreshMembership()
{
    using(new Busy(this))
    {
        var task = AzureService.Instance.GetMembershipById(MembershipId, true);
        await RunSafe(task);

        if(task.IsFaulted)
            return;
    }
}
```

Rob's Tips and Tricks

- When creating UI with C#, avoid unnecessary layout passes by adding the root container to the visual tree at the end
- Create a single, static, reusable instance of Converters
- Unsubscribe from all `MessagingCenter` notifications
- Unwire all your events
  - Consider wiring up during `OnAppearing` and unwiring on `OnDisappearing`
- Add some debug logging to your destructors so you know they are being cleaned up

Xamarin

EVOLVE 16

April 24-28 Orlando, FL

evolve.xamarin.com