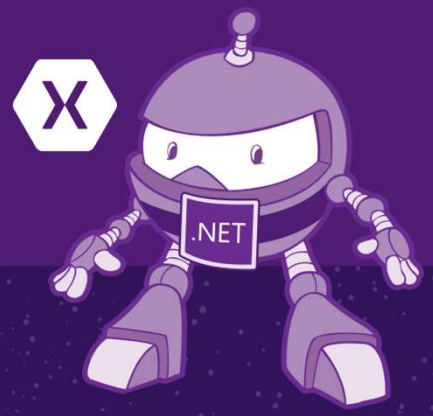

.NET Conf
"Focus on Xamarin"
.NET
focus.dotnetconf.net

Developing Performant Xamarin Apps

Sweeky Satpathy
@SweekritiS



Hot Tips

Brand New App? Be Smart at the Start

Smart Architecture Choices
- Use MVVM
- Repository Pattern for DB

Xamarin.Forms Shell? Yes, if possible*

Pick your dependencies wisely

Test often -> Fix often!

Already have an app?

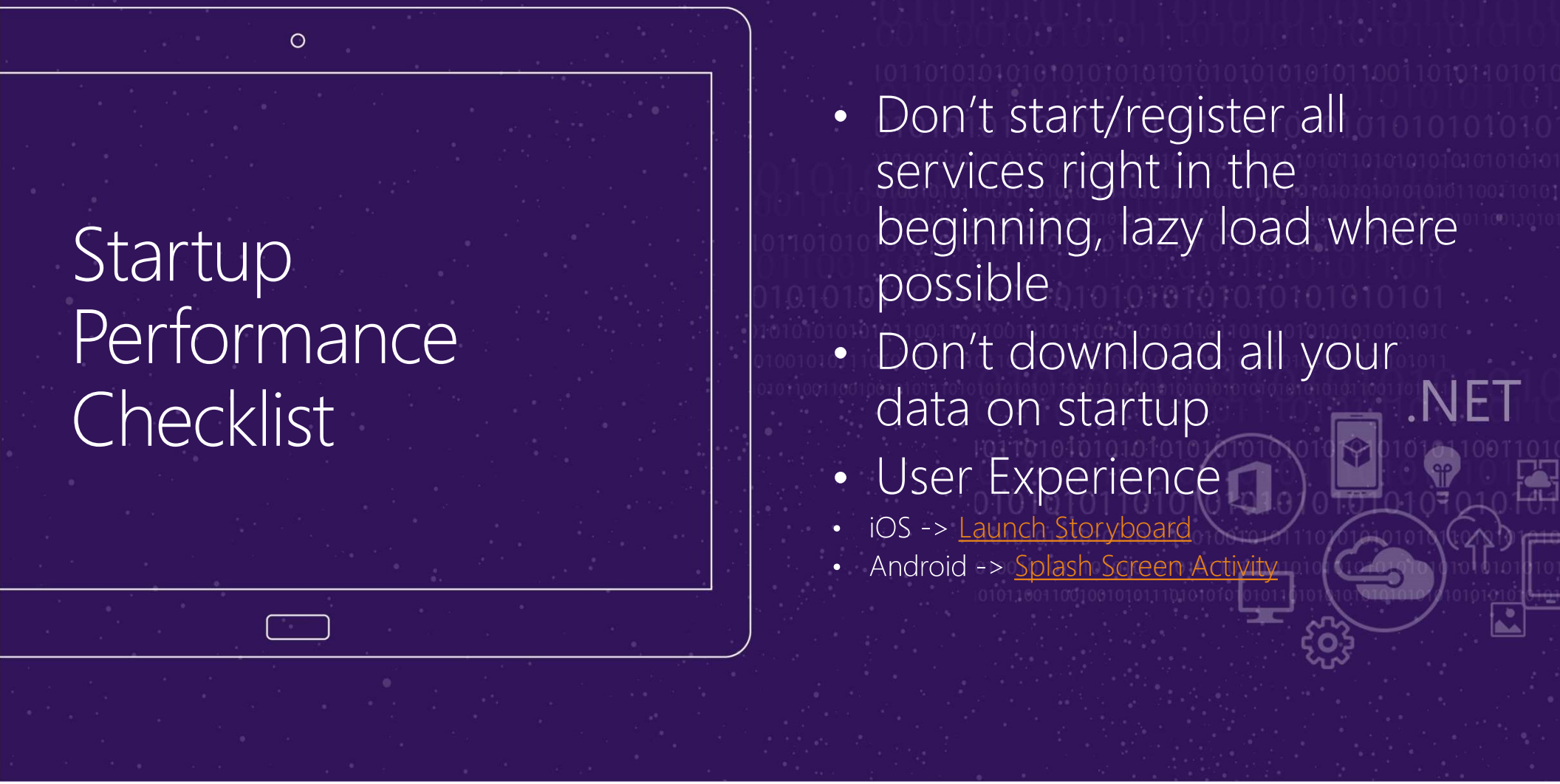
- Checkout this doc : Improve Xamarin.Forms App Performance

Performance Optimizations

.NET

Startup Performance Checklist

- Don’t start/register all services right in the beginning, lazy load where possible
- Don’t download all your data on startup
- User Experience
- iOS -> Launch Storyboard
- Android -> Splash Screen Activity



Android Startup Tracing

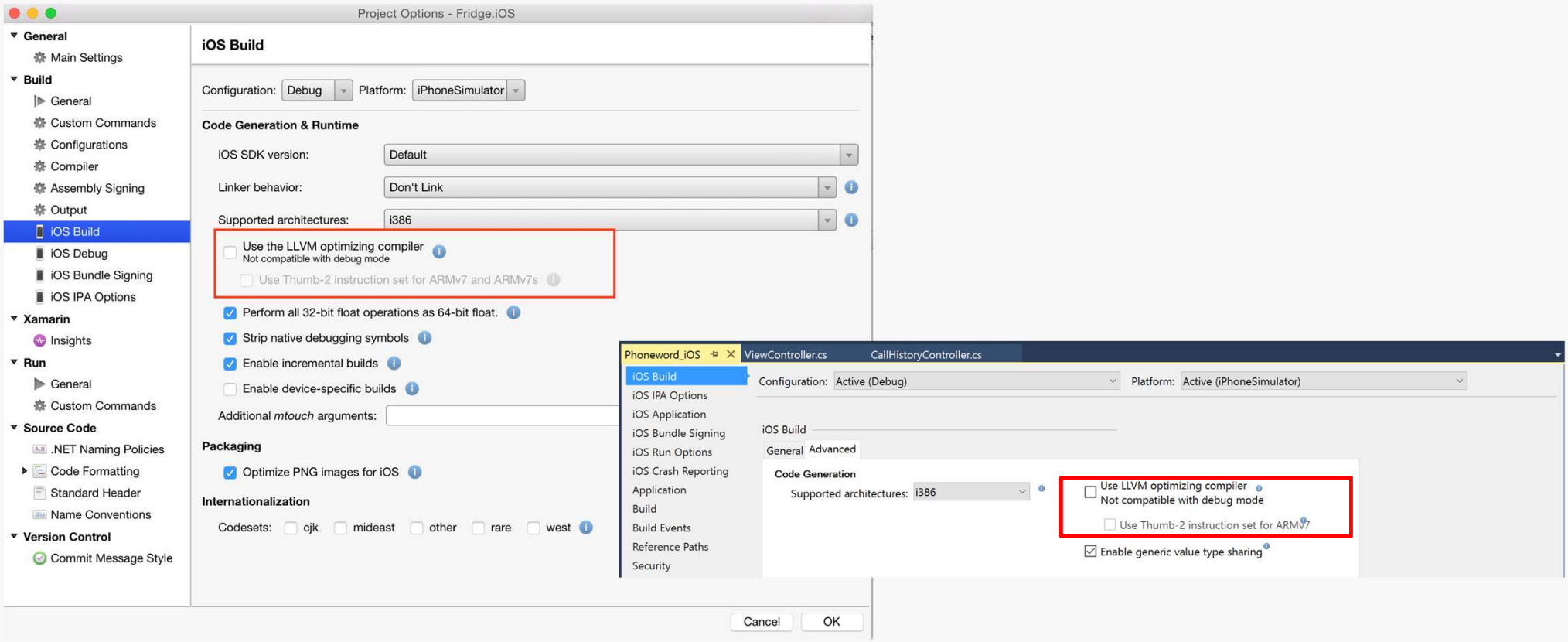
- Use AOT with startup tracking on Android for faster startup
- *NEW* : Faster Application Startup using Custom Profiles with Startup Tracing on Android

You can see the differences in startup tracing modes against a larger Xamarin.Forms application. The measurements below were collected against the SmartHotel360 repository.

	Release	Release/AOT	Release/Startup Tracing (Default)	Release/Startup Tracing(Custom)
Activity Displayed	4863ms	2086ms	3655ms	1900ms
APK Size	48MB	95MB	57MB	60MB

Custom profiles has the most optimal startup performance with a minimal overhead to APK size!

Enable LLVM optimizing compiler for Release configurations - iOS



Other Gotchas

Use Async/Await with Best Practices

- Executing operations consecutively when it is possible to do so concurrently
- Unnecessarily waiting for all Tasks to complete before processing results
- Don’t use Task.Run inside a loop
- Not using cancellation tokens

Consecutive vs Concurrent Operations

Okay Code

```
2 references
async Task<int> ProcessURLAsync(string url)
{
    var content = await Client.GetByteArrayAsync(url).ConfigureAwait(false);
    return content.Length;
}

0 references
async Task<int> SumPagesSizesAsync(List<string> pageUrls)
{
    var total = 0;

    foreach (var url in pageUrls)
        total += await ProcessURLAsync(url).ConfigureAwait(false);

    return total;
}
```

Better Code

```
0 references
async Task<int> SumPagesSizesAsync(List<string> pageUrls)
{
    var lengths = await Task.WhenAll(pageUrls
                                .Select(i => ProcessURLAsync(i)))
                                .ConfigureAwait(false);

    return lengths.Sum();
}
```

Efficiently process Aysnc Operations

Okay Code

```
0 references
async Task<IEnumerable<MyType>> GetItemsAsync(List<string> idValues)
{
    List<MyType> items = new List<MyType>();
    string api_endpoint = "${api_endpoint}/{i}";

    var itemsJson = await Task.WhenAll(idValues
                              .Select(i => Client.GetStringAsync(api_endpoint)))
                              .ConfigureAwait(false);

    foreach (var json in itemsJson)
        items.Add(JsonConvert.DeserializeObject<MyType>(json));

    return items;
}
```

Better Code

```
0 references
async Task<IEnumerable<MyType>> GetItemsAsync(List<string> idValues)
{
    List<MyType> items = new List<MyType>();
    string api_endpoint = "${api_endpoint}/{i}";

    var tasks = idValues.Select(i => Client.GetStringAsync(api_endpoint));

    while (tasks.Count() > 0)
    {
        var completedTask = await Task.WhenAny(tasks)
                                      .ConfigureAwait(false);

        tasks.Remove(completedTask);
        var json = await completedTask;
        items.Add(JsonConvert.DeserializeObject<MyType>(json));
    }
    return items;
}
```

Task Cancellation

`CancellationTokenSource`

- Creates cancellation tokens and sends cancellation requests to all copies of that token

`CancellationToken`

- Used by listeners to monitor current token state

```
var cts = new CancellationTokenSource();
var token = cts.Token;

Task.Run(async () => await someLongRunningAction(),
token);

cts.Cancel();
```

Careful: This task may still run!

Ensuring task cancellation

Checking cancelled status is a manual process!

```
var cts = new CancellationTokenSource();
var token = cts.Token;

Task.Run(() => {
    for (int i = 0; i < 3; i++) {
        token.ThrowIfCancellationRequested();
        // Do some more work
    }
}, token);

cts.Cancel();
```

Resources

- James Clancey's Evolve Talk
- David Ortinau's Blog Post to boot Xamarin.Forms startup time
- Dean Faizal's Xamarin Show Episodes :
  - Best Practices Async/Await
  - Advanced Async/Await
  - When to use async void
- Correcting Common Async/Await Mistakes in .NET
- Using Custom AOT Profiles with Xamarin.Android
- Faster Startup Times with Startup Tracing on Android
- Faster Application Startup using Custom Profiles with Startup Tracing on Android
- Async/Await Resources

Memory Management

.NET

Event Handlers.... 😈

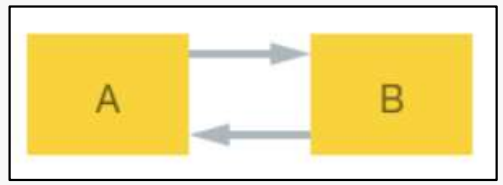
- Always Detach Event Handlers and Dispose Observers
- Unsubscribe from events and avoid anonymous delegates to prevent memory leaks
- A reference to the anonymous method can be stored in a field and used to unsubscribe from the event

```
public class Subscriber : IDisposable
{
  readonly Publisher publisher;
  EventHandler handler;

  public Subscriber (Publisher publish)
  {
    publisher = publish;
    handler = (sender, e) => {
      Debug.WriteLine ("The publisher notified the subscriber of an event");
    };
    publisher.MyEvent += handler;
  }

  public void Dispose ()
  {
    publisher.MyEvent -= handler;
  }
}
```

```
public class Publisher
{
  public event EventHandler MyEvent;

  public void OnMyEventFires ()
  {
    if (MyEvent != null) {
      MyEvent (this, EventArgs.Empty);
    }
  }
}

public class Subscriber : IDisposable
{
  readonly Publisher publisher;

  public Subscriber (Publisher publish)
  {
    publisher = publish;
    publisher.MyEvent += OnMyEventFires;
  }

  void OnMyEventFires (object sender, EventArgs e)
  {
    Debug.WriteLine ("The publisher notified the subscriber of an event");
  }

  public void Dispose ()
  {
    publisher.MyEvent -= OnMyEventFires;
  }
}
```

Use weak references to prevent immortal objects

This diagram illustrates a strong reference creating an unwanted immortal object



WeakReference code example

```
public class A
{
  B b;

  public A ()
  {
    b = new B (this);
  }
}

public class B
{
  readonly WeakReference<A> aWeakRef;

  public B (A a)
  {
    aWeakRef = new WeakReference<A> (a);
  }
}
```

Load Data…Efficiently

Load local content first

- Use placeholder text/images/loading icon
- Minimize your web payload to just what you need

Don't bind things that can be set statically

- Descriptive Labels/Titles don’t need to be bound

Profilers

Profilers! Profilers! Profilers!

Look for

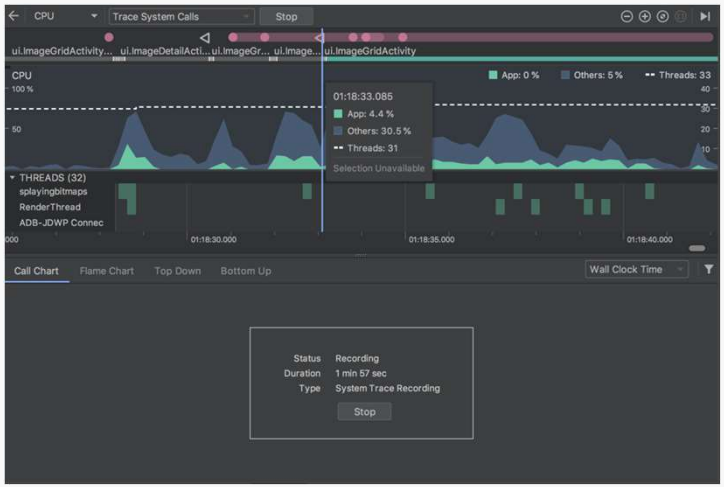
- memory leaks, large images, large objects graphs, wide scopes, cross-references, contexts that prevent the garbage collector from working properly

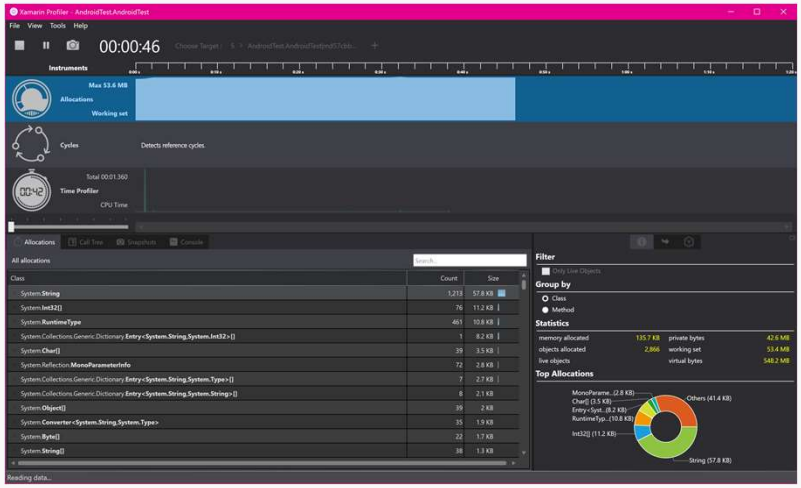
Measure

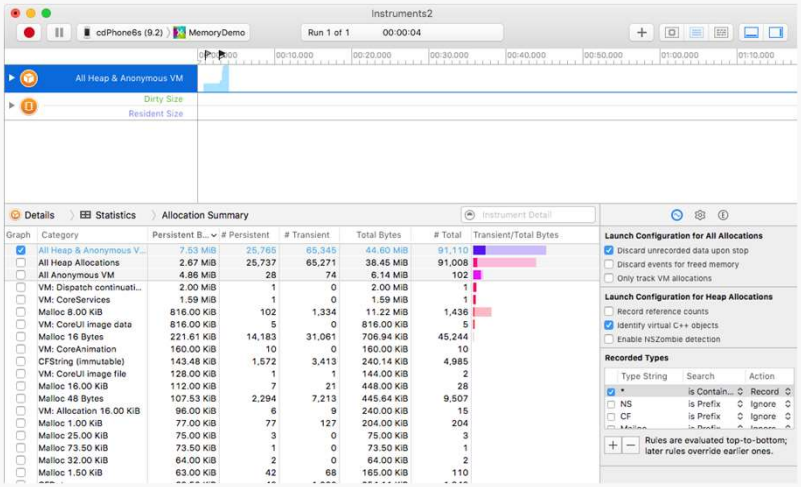
- startup time, operation time, memory consumption, CPU profile, networking profile, I/O profile

Profiler Options

- Xamarin Profiler
- Xcode Instruments
- Android Studio Profiler







Resources

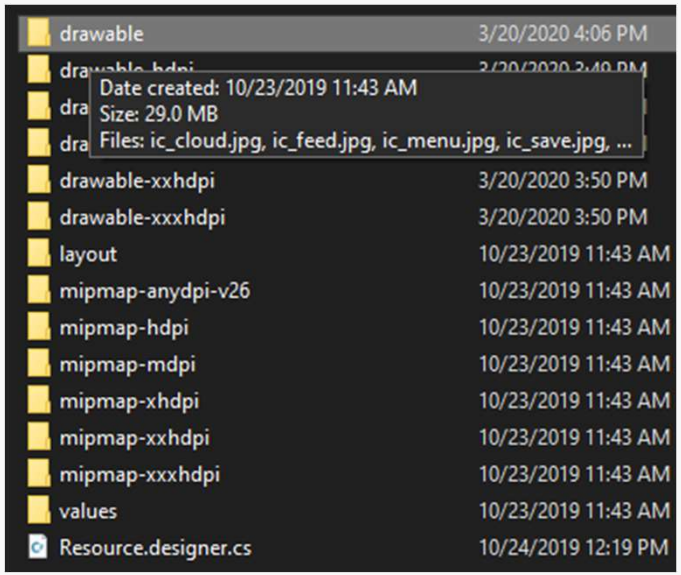
- Xamarin.Forms Memory Performance Best Practices
- Profiling Xamarin.iOS Applications with Instruments
- Profiling Android Apps
- Xamarin.Android Performance
- Alexey Strakh's Xamarin Show Episodes :
  - Memory Management

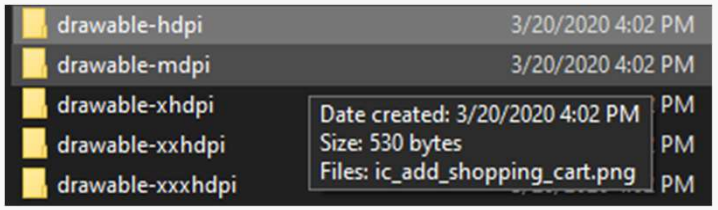
Manage Your Resources

.NET

Icons, Images, Assets

Using unoptimized assets for each platform and form factor





Faster Image Loading

Avoid the following :

- Putting large files in .NET Standard project
- Defer key/fundamental images or resources by loading from the web

Consider using

- GlideX.Forms
- Sharpnado's Xamarin.Forms.Nuke

Implement caching for frequently used images

- FFImageLoading

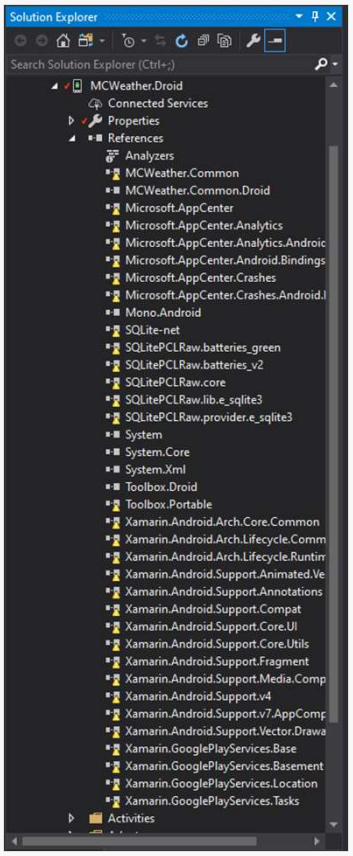
Resources

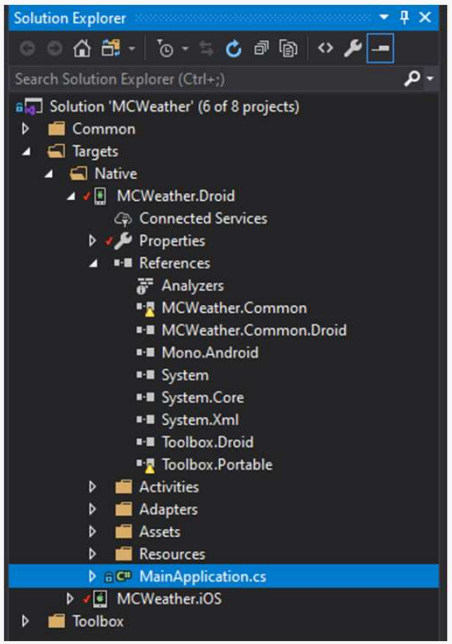
- Jonathon Pepper's Blog Post on GlideX for Android
- Android Asset Studio
- Xamarin.Android Alternate Resources
- Sharpnado's Xamarin.Forms Nuke
- FFImageLoading for Xamarin.Forms
- mFractor plugin for Visual Studio
- Shared Images for Xamarin with Resizetizer NT

Dependencies

Optimize dependencies being used

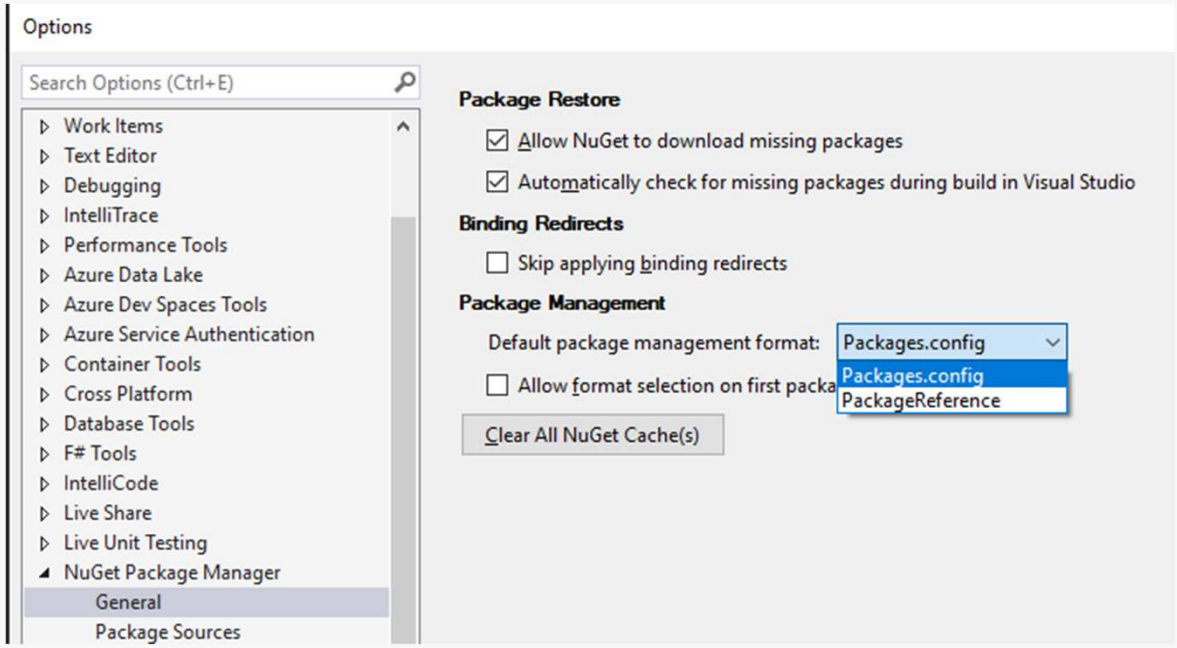
extra nugets, dependencies, remove everything not being used





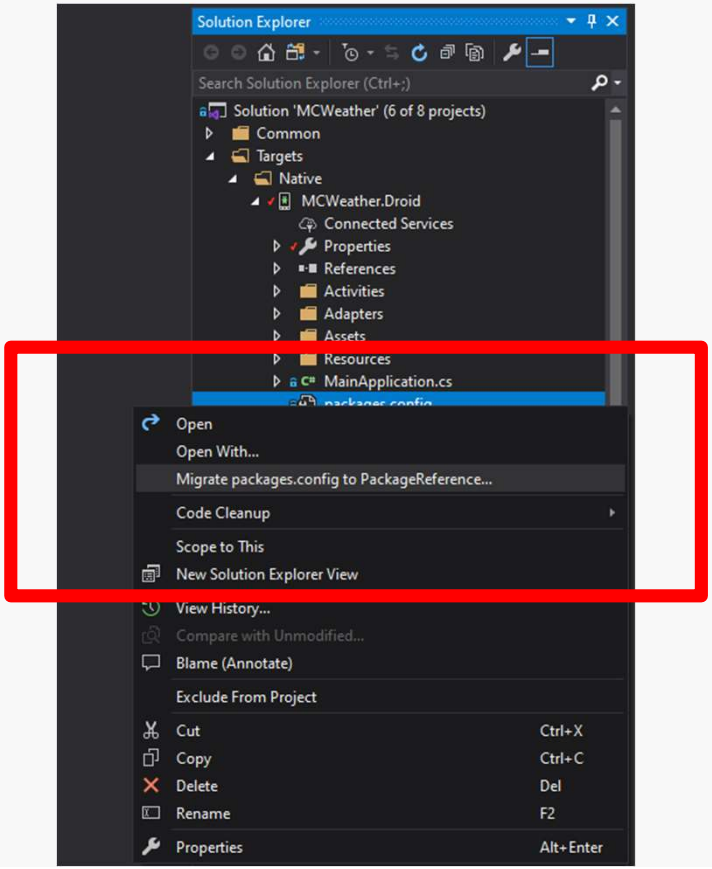
Package Reference

Set your NuGet Package Manager options to use PackageReference as the default!



Quick Migration Tool

Right Click Magic!



Resources

- Jonathan Pepper's Android Performance Guide
- Migrate packages.config to PackageReference

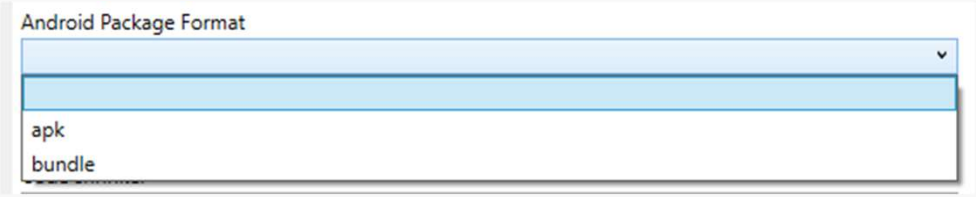
App Packaging

Let's Talk Linker

- Linker Settings :
  - Turn on **Link SDK Assemblies Only** when your app is in **Release [at the very least]**
  - Turn on **Link All/SDK and User Assembles** when your app is in **Release [Test manually]**
- Enable [assembly:LinkerSafe] in binding projects

Android App Bundles

- Optimization of a single APK that contains the necessary resources for all of your target ABI's



Resources

- Investing Time in the Xamarin Linker for Smaller App Sizes
- Optimizing Xamarin Apps & Libraries with the Linker
- Linking on Xamarin.Android
- Linking on Xamarin.iOS
- Xamarin.Android Linker Tricks Part 1
- Xamarin.Forms performance on Android
- Optimize Xamarin.Android builds
- Xamarin Android App Bundles
- Xamarin Show - Android App Bundles

UI Tips

Good Layout Tips

- Avoid Nested Layouts
- Don’t use “Auto” in Grids
- Use DataTemplates in ListView/CollectionView
- [Android] Use AppCompat

```
<StackLayout>
    <StackLayout>
        <StackLayout>
            <Label>...text...</Label>
        </StackLayout>
        <StackLayout>
            <Label>...text...</Label>
        </StackLayout>
    </StackLayout>
    <StackLayout>
        <StackLayout>
            <Label>...text...</Label>
        </StackLayout>
        <StackLayout>
            <Label>...text...</Label>
        </StackLayout>
    </StackLayout>
</StackLayout>
```

Fonts

- Use font general names instead of measuring and setting numbers
- Unnecessarily using graphics for assets -> Use Font for Icons

```
<Style TargetType="Tab" x:Key="FollowTab">
    <Style.Triggers>
    <Trigger TargetType="Tab"
             Property="IsChecked" Value="False">
        <Setter Property="Icon" >
            <Setter.Value>
                <FontImageSource FontFamily="{StaticResource FontAwesomeRegular}" Glyph="&#xf004;"/>
            </Setter.Value>
        </Setter>
    </Trigger>
    <Trigger TargetType="Tab"
             Property="IsChecked" Value="True">
        <Setter Property="Icon" >
            <Setter.Value>
                <FontImageSource FontFamily="{StaticResource FontAwesomeSolid}" Glyph="&#xf004;"/>
            </Setter.Value>
        </Setter>
    </Trigger>
    </Style.Triggers>
</Style>
```

Updating Views

- Be careful about updating non-visible Tabbed Pages

Resources

- Choose Correct Layout
- Toggling Tabs with Triggers
- Dynamically Changing Xamarin.Forms Tab Icons When Selected
- MobCAT - XamTwitch Repo

Questions!

Thank You!
Wash your hands!

Sweeky Satpathy
@SweekritiS