

# Debugging Vampires

AKA: Where did the reflection go?

Also some other stuff about  
building projects for Windows RT

## WinRT - The Windows Runtime

- Striped down version of core microsoft libs; System.dll, mscorlib.dll etc.
- Unity **does not** compile against WinRT in the editor, only when building.
  - Compiler will ignore code in /Plugins/
- Patching functions missing from WinRT is your main task when porting to Windows mobile.

## Case Study: LitJson

- Uses reflection to convert `Json<->Object`
- Compiled DLL appears to work...
  - But you will fail WACK (More on this later).
- Luckily we have the source (It's public domain!).
  - [github.com/lbv/litjson](https://github.com/lbv/litjson)

# Case Study: LitJson

- **Error Found:** The supported APIs test detected the following errors:

- API System.Type.GetInterface(System.String) in MSCORLIB, PUBLICKEYTOKEN=B77A5C561934E089 is not supported for this application type. LitJson.dll calls this API.
- API System.IO.TextReader.Close in MSCORLIB, PUBLICKEYTOKEN=B77A5C561934E089 is not supported for this application type. LitJson.dll calls this API.

## Development Console

```
Exception: Method not found: 'System.Reflection.FieldInfo[]  
System.Type.GetFields(System.Reflection.BindingFlags)'.  
Type: System.MissingMethodException  
Module: Assembly-CSharp  
InnerException: <No Data>  
AdditionalInfo:<No Data>  
  at GameManager.Awake()  
  at GameManager.UnityFastInvoke_Awake()
```

DLL's in /plugins/  
throw **Method Not Found Exceptions** if  
unsupported.

# Case Study: LitJson

- Assets\Scripts\LitJson\ error CS1061: 'System.Type' does not contain a definition for 'GetInterface' and no
- Assets\Scripts\LitJson\ error CS1061: 'System.Type' does not contain a definition for 'GetProperties' and n
- Assets\Scripts\LitJson\ error CS1061: 'System.Type' does not contain a definition for 'GetFields' and no ex
- Assets\Scripts\LitJson\ error CS1061: 'System.Type' does not contain a definition for 'GetProperties' and n
- Assets\Scripts\LitJson\ error CS1061: 'System.Type' does not contain a definition for 'GetFields' and no ex
- Assets\Scripts\LitJson\ error CS1061: 'System.Type' does not contain a definition for 'GetMethod' and no e
- Assets\Scripts\LitJson\ error CS1061: 'System.Type' does not contain a definition for 'IsClass' and no exte
- Assets\Scripts\LitJson\ error CS1061: 'System.Type' does not contain a definition for 'IsAssignableFrom' a
- Assets\Scripts\LitJson\ error CS1061: 'System.Type' does not contain a definition for 'IsEnum' and no exte
- Assets\Scripts\LitJson\ error CS1061: 'System.IO.TextReader' does not contain a definition for 'Close' and

## Case Study: LitJson

- Reflection paired down in WinRT, probably to enforce the app sandbox (Speculation).
- We can hardcode our changes by editing LitJson itself... but it's bad practice.
- Use extension methods instead! Make direct edits only when necessary.

# Case Study: LitJson

```
#if !UNITY_EDITOR && UNITY_METRO
namespace LitJson {
    public static class WinRTPatch {
        // Extension methods allow us to extend classes we may not
        // have access to by using the 'this' keyword.
        public static PropertyInfo[] GetProperties(this Type _type){
            return _type.GetRuntimeProperties().ToArray();
        }
    }
}
#endif
```

# Case Study: LitJson

Extension Methods:

- Modify classes non-destructively.
- Great for compatibility patches.
- Patch could be released as standalone fix.
- More on extensions:  
[wikipedia.org/wiki/Extension\\_method](http://wikipedia.org/wiki/Extension_method)  
[MSDN/library/bb383977.aspx](http://MSDN/library/bb383977.aspx)



## Case Study: LitJson

- WP8 has entirely different libs, similar problems will occur (Use #directives!).
- Watch out for function overrides:  
GetInterface(name) - Unsupported  
GetInterface(name,ignoreCase) - Supported
- 4.3 compiler much better than in 4.2, will catch most problems at the Unity build step.

# Building for WinRT

Building for WinRT is a 2 step process:

1. Unity generates Visual Studio project.
  - Use 'XAML C# Solution' & 8.1 settings.
  - 8.0 apps no longer supported by Microsoft.
2. Visual Studio Project edited and compiled.
  - Add settings flyout/other required funcs.
  - Localization for store info.
  - Build .appx for testing or publishing.

# Unity <-> WinRT interop

- In order to comply with Microsoft guidelines, we need to implement some Metro functionality.
- Settings Flyout, Privacy Policy, Window snap & resize.
- C# compiled by Unity can be directly accessed from Metro, makes integration fairly simple. But **Unity is not thread safe!** Watch out!

# Unity <-> WinRT interop

```
// This class goes in the generated visual studio project!
public class AppSettings : SettingsFlyout
{
    private void flyoutMute_Toggled(object sender, RoutedEventArgs e){
        // Call functions with InvokeOnAppThread to avoid
        // threading crashes.
        AppCallbacks.Instance.InvokeOnAppThread(() => {
            // Use events if possible, extra guard against threading.
            MySoundController.RaiseMuteToggleEvent();
        }, false);
    }
}
```

# Unity <-> WinRT interop

- Windows App guidelines:  
<http://msdn.microsoft.com/en-us/library/windows/apps/hh694083.aspx>
- WinRT interop code included in our starter kit.
- Excellent projects & examples on the MSDN.
- Better Unity integration in the future?

# Freemium API

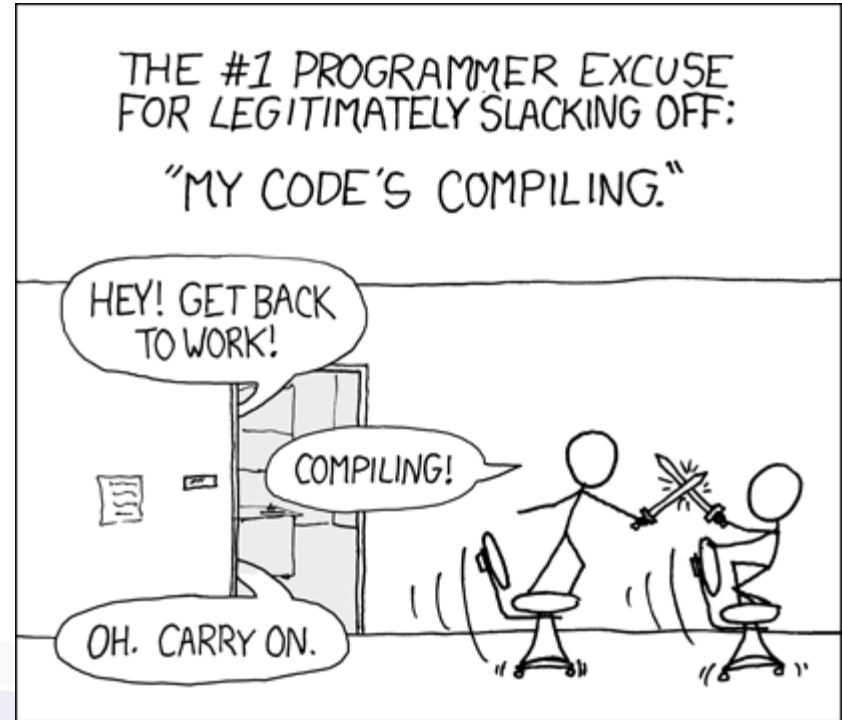
- Microsoft has 'Trial Period' option that will disable access to your app after a period of time.
- Entirely managed by Microsoft! Lines of code I had to write: Zero -- Awesome!
- Lightweight API for querying trial time left, not required though.

# Windows App Cert Kit - WACK

- Scans your app automatically and will highlight problem areas in code and in performance.
- Is the first step in the store cert process. **Failing WACK means you will fail store cert, use it!**
- Most common WACK failures: Unsupported API & long startup time.

# Windows App Cert Kit - WACK

- Comes with VS2013 for Windows 8.
- Separate exe for WinRT devices. **Test on both!**
- Takes about 20-30 min, depending on app size.



xkcd.com / R. Munroe | CC BY-NC 2.5



# Notes on Compiling

'Compilation Overrides' Unity PlayerSetting:

- 'None' - Don't use this, seems like it solves all of your missing method problems... but you will fail WACK.
- 'Use Net Core' - Default, compiles for WinRT.
- 'Use Net Core Partially' - Allows C# $\leftrightarrow$ JS/Boo interop. WACK compatible.

# Notes on Compiling

Visual Studio Compile Modes:

- 'Debug' - Compile for debugging.
- 'Release' - ???, Leaves 'Development Build' watermark in build. Just don't use it.
- 'Master' - Compile for release to the app store.

# Notes on Compiling

Overwriting an existing VS project.

- Unity will only replace the /Data/ folder, everything else will be unchanged.
- DLLs added to the VS project by Unity **will not be removed by Unity** if you delete them from the Unity Project, you must manually delete them or create a fresh VS project.



# Odds and Ends

- Unity runtime log:  
`<USER>\AppData\Local\Packages\<Project Name>\TempState`
- WP8 devs: check out 'Windows Phone Power Tools' for retrieving the runtime log.
- Powershell: .ps file generated with .appx, used to deploy test builds.

## Odds and Ends

Navigating the MSDN:

- Look for the briefcase icon.
- Check the Version Information.

	IsGenericParameter
	IsGenericType

### ▲ Version Information

**.NET Framework**

Supported in: 4.5.1, 4.5, 4, 3.5, 3.0, 2.0

**.NET Framework Client Profile**

Supported in: 4, 3.5 SP1

**Portable Class Library**

Supported in: Portable Class Library

**.NET for Windows Store apps**

Supported in: Windows 8

# Thanks!

- Project with all code discussed available on bitbucket:  
[bitbucket.org/Mervill/winrt-starter-kit](https://bitbucket.org/Mervill/winrt-starter-kit)  
Format: Mercurial/7z  
Licensed under MIT/Boost, use it to kickstart your WinRT project!
- [riley@boximals.com](mailto:riley@boximals.com)  
/in/rileygodard  
@Mervill\_