MSK
DOT
NET

Migrating server apps from the .NET Framework to .NET Core

Raffaele Rialdi
Senior Software Architect
Microsoft MVP

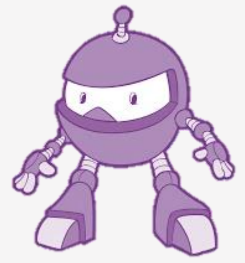
@raffaeler
https://github.com/raffaeler
http://iamraf.net

Who am I?

- Raffaele Rialdi, Senior Software Architect in Vevy Europe – Italy
  @raffaeler also known as "Raf"
- Consultant in many industries
  Manufacturing, racing, healthcare, financial, …
- Speaker and Trainer around the globe (development and security)
  Italy, Romania, Bulgaria, Russia (Moscow, St Petersburg and Novosibirsk), USA, …
- And proud member of the great Microsoft MVP family since 2003

La sessione non è:
….. un tutorial su come hackerare sistemi
….. una dimostrazione sulle vulnerabilità di Windows
….. adatta ai deboli di cuore

What is Net Core



- Technically a «fork» of the .NET Framework
  - Looks the same CLR and Base Class Library but greatly improved
  - It works Cross Platform (Linux, Mac, Windows) + x86, x64, ARM
- Scenarios taking advantage from Net Core (currently) are:
  - ASP.NET Core: new ASP.NET stack, re-written from scratch
  - Universal Windows Platform (Windows Store Apps)
  - Cloud Applications: Applications and Microservices running on Azure
  - Console Application: the best way to start testing Net Core
- The best choice for containerized applications / docker ready
- +1 Million new monthly .NET active developers in 2017-2018

Which libraries work in .NET Core?

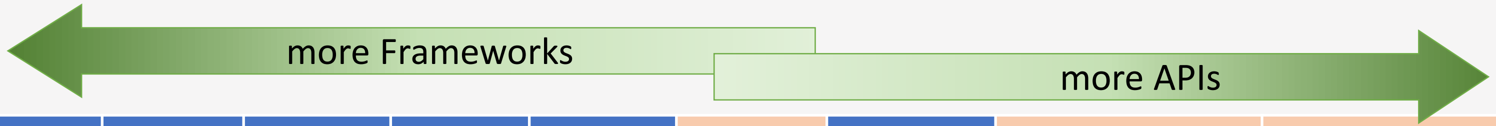
- No UI stack (desktop apps). Very soon with .NET Core 3.0!
    - UWP apps already use .NET Core with .NET Native toolchain
- ASP.NET Core is totally re-written
    - Very similar concepts but much faster and pluggable
- OData version 7.0+ now runs on both ASP.NET and ASP.NET Core
- Entity Framework Core is totally re-written
    - EF6 will be ported in .NET Core 3.0 only to ease the migration of apps
- SignalR Core has been totally re-written
- New platform-independent logging system
    - ETW on Windows, LTTNG on Linux
- Identity framework has major (breaking) changes

What is netstandard

https://github.com/dotnet/standard/

- A library specification defining a set of APIs with no implementation
  - Think to netstandard as it was a huge interface
  - All NET Frameworks must implement all those APIs
- CoreFX is a superset of netstandard
  - https://github.com/dotnet/corefx
- The review board defines the number of netstandard APIs
  - No platform specific APIs but abstractions to access to the OS services
  - Only APIs that should be always available
- Netstandard avoids the confusion of external libraries
  - NodeJS suffers from this problem

netstandard versions vs Platforms



.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0	2.1
.NET Core							1.0	2.0	3.0
.NET Framework		4.5	4.5.1	4.6	4.6.1	4.6.1	4.6.1	4.6.1 §	**N/A**
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4	vNext
Xamarin.iOS	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.14	vNext
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.8	vNext
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0	vNext
UWP	10.0	10.0	10.0	10.0	10.0	10.0. 16299	10.0. 16299	10.0. 16299	vNext

§ before 4.7.2 lot of nuget packages are required to fill the gap with netstandard 2.0

Unity Framework ➔ netstandard 2.0

Version	**# of APIs**
netstandard 1.6	13,501
netstandard 2.0	32,638
netstandard 2.1	35,742

netstandard 2.0 vs 2.1

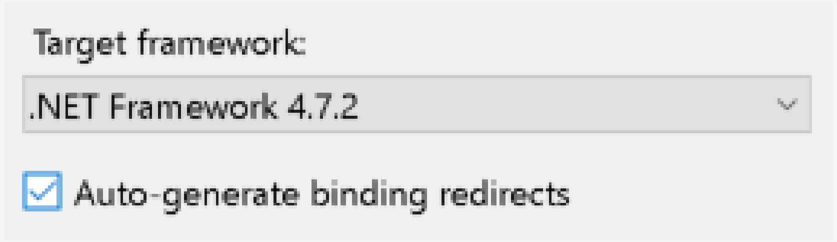
- Added in 2.1
  - Span<T>, Memory<T>, MemoryExtensions, MemoryMarshal, …
    - And the ReadOnly counterparts
  - Tons of new overloads taking Span<T> across the whole corefx
  - Reflection.Emit
  - Vector<T> (SIMD support)
  - ValueTask and ValueTask<T>
  - DbProviderFactories
- .NET 4.8 will stay on netstandard 2.0 !
- .NET Core 3.0, Xamarin, Mono, and Unity will get netstandard 2.1

https://github.com/dotnet/standard/blob/master/docs/versions/netstandard2.1_diff.md

Migrating a codebase

The typical migration process

1. Migrate to .NET Framework 4.7.2
   - First version to fully support netstandard 2.0 without additional nuget packages or tooling support
   - CLR has been modified to avoid the need of app.config bindingRedirect



Target framework:

.NET Framework 4.7.2

[✓] Auto-generate binding redirects

2. Convert libraries to netstandard 2.0
   - Start from the bottom of the dependency graph
3. Exe and web apps still target .NET Framework 4.7.2

Migrating tweaks

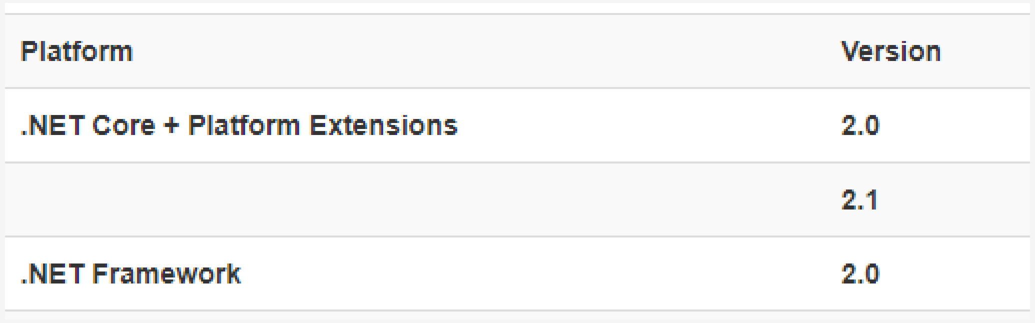
- Use Dependency Injection
  - Adopt a D.I. framework to lazy load framework specific libraries
  - Autofac is a very popular example
- If a library does not support netstandard, it can be loaded via D.I.
  - This cuts the developer dependency
  - For example libraries using Entity Framework 6
- Nuget is your friend (we will see why in a moment)

Migrating a real ASP.NET Core application

- Write custom middleware to access the raw http requests/responses
  - no more custom http/routing extensibility required
- Migrate the ASP.NET Application
  - use the power of the new ASP.NET Core features (IoC, security, …)
- Add a dockerfile to run inside a container

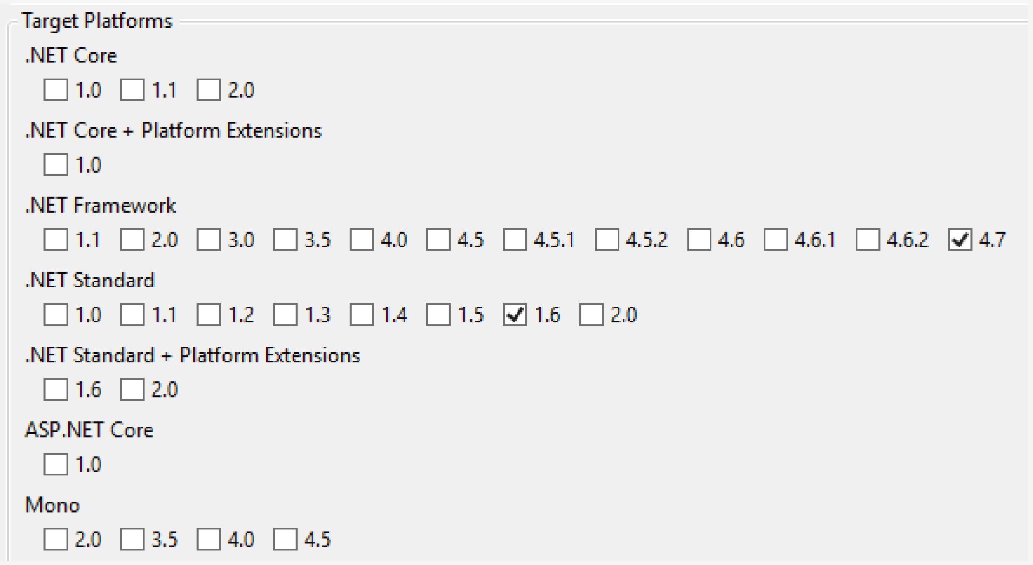
Tools helping migration

- .Net API Catalog http://apisof.net
  - An exhaustive catalog of Microsoft and 3rd parties APIs with framework versions



Platform	Version
.NET Core + Platform Extensions	2.0
	2.1
.NET Framework	2.0

- .Net Portability Analyzer
  - Available in VS2017 and console
  - Generates a report listing the APIs that are not available in the selected framework



Target Platforms

.NET Core
☐ 1.0 ☐ 1.1 ☐ 2.0

.NET Core + Platform Extensions
☐ 1.0

.NET Framework
☐ 1.1 ☐ 2.0 ☐ 3.0 ☐ 3.5 ☐ 4.0 ☐ 4.5 ☐ 4.5.1 ☐ 4.5.2 ☐ 4.6 ☐ 4.6.1 ☐ 4.6.2 ☑ 4.7

.NET Standard
☐ 1.0 ☐ 1.1 ☐ 1.2 ☐ 1.3 ☐ 1.4 ☐ 1.5 ☑ 1.6 ☐ 2.0

.NET Standard + Platform Extensions
☐ 1.6 ☐ 2.0

ASP.NET Core
☐ 1.0

Mono
☐ 2.0 ☐ 3.5 ☐ 4.0 ☐ 4.5

Avoid AppDomains

- In NetCore there is just one AppDomain per process
  - The CreateDomain API exists but throws
- Are you using them for reflection only?
  - The System.Reflection.Metadata nuget package is for you
    - Read the raw ECMA-335 metadata reader but you have a lot of work to do then
- Do you need them for a single one-shot execution?
  - Run a process and interop with stdin/stdout
- Do you need them for multiple requests/responses?
  - Use HTTP on localhost using ASP.NET Core

Avoid …

- Windows Registry
  - Certain calls on Linux throws :)

- Using directly ETW
  - The new Microsoft logger infrastructure wraps ETW and LTTNG

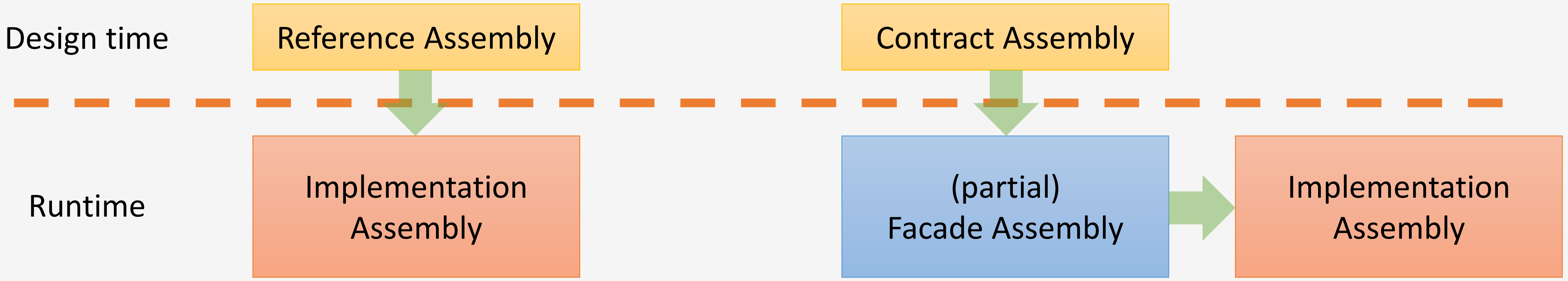
NuGet and Reference Assemblies

Type forwarding

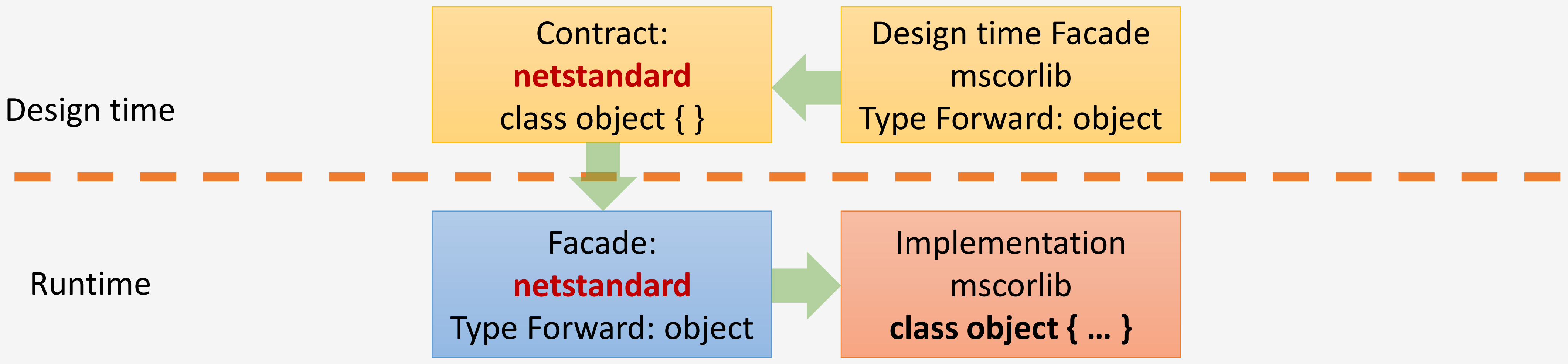
- It is an entry in an "Assembly Exported Type Table" (assembly metadata) containing a "redirection" to a type in another assembly
- Used when you want <u>move a type</u> from an assembly to another one

- The "Facade Assembly" are assemblies containing <u>just</u> type forwards
  - They contain no IL code, or metadata other than "assembly metadata"
- The "Partial Facade Assembly" are "Facade Assembly" containing type definitions too.

Reference and Contract assemblies



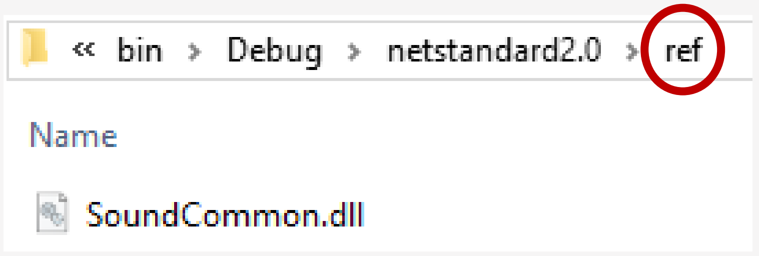
How System.Object gets 'redirected' on netstandard



Generating our own Reference Assemblies

- Why do we want generating reference assemblies?
  - When generating libraries used by other developers (smaller dependencies)
    - You must include every reference to types that are part of the public signatures
  - Providing multiple implementations of the same public API
    - X-plat, X-framework, test scenarios, …
- How do we generate reference assemblies?
  - Modifying the csproj

```xml
<PropertyGroup>
  <Deterministic>true</Deterministic>
  <ProduceReferenceAssembly>true</ProduceReferenceAssembly>
</PropertyGroup>
```



« bin › Debug › netstandard2.0 › ref

Name

SoundCommon.dll

or

  - EmitOptions from Roslyn API

The ProcessInfo Demo

- The process id is platform specific
  - Windows: GetCurrentProcessId (Kernel32.dll)
  - Linux: getpid (libc.so.6)

- Demo goal: creating a **single** nuget package providing the required implementation for the desired runtime

First proposal (please don't use it!)

```
public static class ProcessInfo
{
#if Windows
        [DllImport("Kernel32.dll", EntryPoint="GetProcessId")]
        private static extern int ReadProcessId();

#elif Linux
        [DllImport("libc.so.6", EntryPoint="getpid")]
        private static extern int ReadProcessId();

#else
#error "You must specify 'Windows' or 'Linux'!"
#endif

    public static int GetId() => ReadProcessId();
}
```

Second chance, creating a reference assembly

- Here is the code inside the reference assembly:
  - public declarations only
  - just the signatures, no implementation provided

```
public static class ProcessInfo
{
    public static int GetId() => throw null;

    public static string Name => throw null;
}
```

The platform specific implementations

Windows Specific Assembly

```csharp
public static class ProcessInfo
{
    [DllImport("Kernel32.dll",
         EntryPoint = "GetCurrentProcessId")]
    private static extern int ReadProcessId();

    public static int GetId() =>
        ReadProcessId();

    public static string Name
    {
        get => $"Windows: Pid {GetId()}";
    }
}
```

Linux Specific Assembly

```csharp
public static class ProcessInfo
{
    [DllImport("libc.so.6",
         EntryPoint = "getpid")]
    private static extern int ReadProcessId();

    public static int GetId() =>
        ReadProcessId();

    public static string Name
    {
        get => $"Linux: Pid {GetId()}";
    }
}
```

Implementation assemblies must have the same name!

Packaging the cross-platform ProcessInfo

- The nuspec file must specify the correct sections for each file
  - Reference assembly

```
<file src="ProcessInfo\bin\Debug\netstandard2.0\ProcessInfo.dll"
        target="ref\netstandard2.0\ProcessInfo.dll" />
```

  - Windows assembly

```
<file src="ProcessInfoWindows\bin\Debug\netstandard2.0\ProcessInfo.dll"
    target="runtimes\win\lib\netstandard2.0\ProcessInfo.dll" />
```

  - Linux assembly

```
<file src="ProcessInfoLinux\bin\Debug\netstandard2.0\ProcessInfo.dll"
        target="runtimes\linux\lib\netstandard2.0\ProcessInfo.dll" />
```

To sum up

- In most of the cases, the migration is pretty easy
- Use the compatibility tool and apisof.net website
- Entity Framework 6 migration takes more time
- .NET Core is the future! You will not regret ☺

Questions?

```
var memory = new byte[1024].AsMemory;
var span = memory.Span;
```

Thank you!