.NET: лечение зависимостей

DOTNEXT

Евгений Пешков
JetBrains
telegram/twitter: @epeshk

Dependency hell (DLL Hell)

- Breaking changes (source & binary) – невозможность использовать зависимость из-за изменений в её коде
- Решение 1:
  - Сохранять обратную совместимость

https://docs.microsoft.com/en-us/dotnet/standard/library-guidance/breaking-changes

Breaking changes

```
void Method() -> void Method(int parameter = 0)
```

- Код продолжает компилироваться
- Но бинарно – это разные методы

- Решение 2:
  - Версионировать зависимости
    - `Library.dll, Version=1.0.0.0`
    - `Library.dll, Version=2.0.0.0`

https://docs.microsoft.com/en-us/dotnet/standard/library-guidance/breaking-changes

Новые проблемы

Version hell – невозможность использовать совместимую зависимость из-за правил версионирования, даже если они обратно совместимы

```
System.IO.FileLoadException: Could not load file or assembly 'Newtonsoft.Json, Version=6.0.0.0, Culture=neutral, PublicKeyToken=30ad4fe6b2a6aeed' or one of its dependencies. The located assembly's manifest definition does not match the assembly reference.
```

Виды версий

System.Collections.Immutable

- NuGet Package Version: 1.6.0
- Assembly Version: 1.2.4.0
- Assembly File Version: 4.700.19.46214
- Assembly Information Version: 3.0.0+4ac4c036...

- lib/netstandard**1.0**/System.Collections.Immutable.dll
- lib/netstandard**1.3**/System.Collections.Immutable.dll
- lib/netstandard**2.0**/System.Collections.Immutable.dll

Документация

- .NET Guide – 328 pages PDF
  - Assemblies in .NET – 87 pages
  - Open source library guidance – 26 pages

https://docs.microsoft.com/en-us/dotnet/standard/
https://docs.microsoft.com/en-us/dotnet/opbuildpdf/standard/toc.pdf?branch=live

Для чего это нужно?

- Избежать граблей с зависимостями

Для чего это нужно?

- Избежать граблей с зависимостями
- Сделать жизнь пользователей ваших библиотек проще
- Справиться с проблемами при миграции на .NET Core
- Стать SRE – Senior (Binding) Redirect Engineer

О чём будем говорить

- Strict assembly loading
  - Binding redirects
  - Strong naming
- .NET Core
  - Shared frameworks, .runtimeconfig.json
  - Dependency manifest (.deps.json)
  - Хаки для запуск JetBrains Rider на Core
- Отладка загрузки сборок
  - Fusion logs
  - Runtime events

1. Strict assembly loading

Main .NET Framework assembly loading issue

Basics

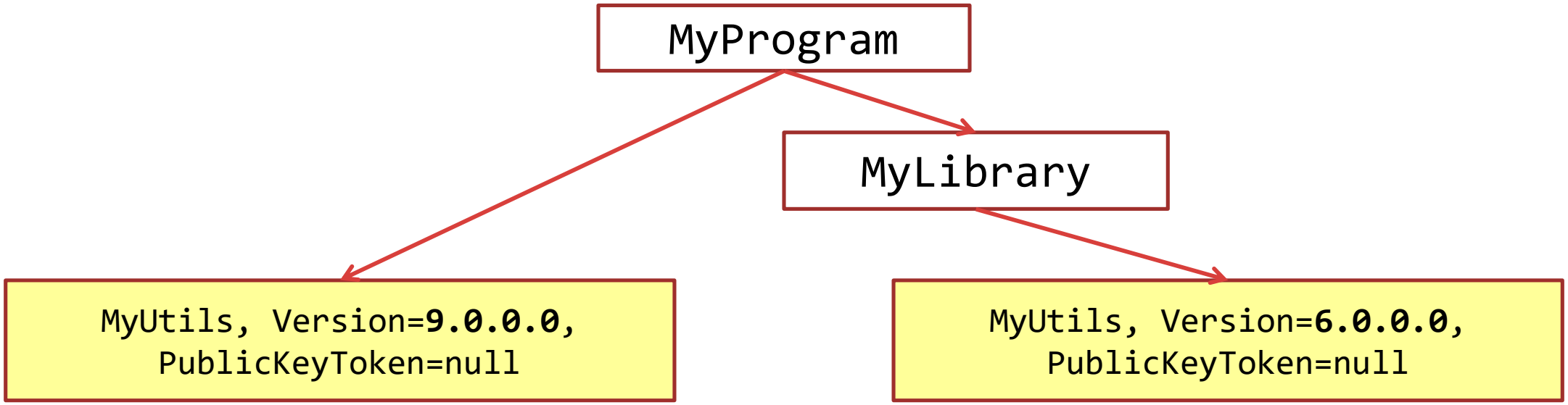
- Артефакт сборки проекта (BIN) – набор сборок и конфигурационных файлов
- Каждая сборка внутри себя содержит ссылки на другие сборки по Assembly Name
- Assembly resolving
  - Design time
  - Runtime

Assembly Name kinds

```
// Simple name
MyAssembly, Version=6.0.0.0,
Culture=neutral, PublicKeyToken=null

// Strong name
Newtonsoft.Json, Version=6.0.0.0,
Culture=neutral, PublicKeyToken=30ad4fe6b2a6aeed // PublicKey
```

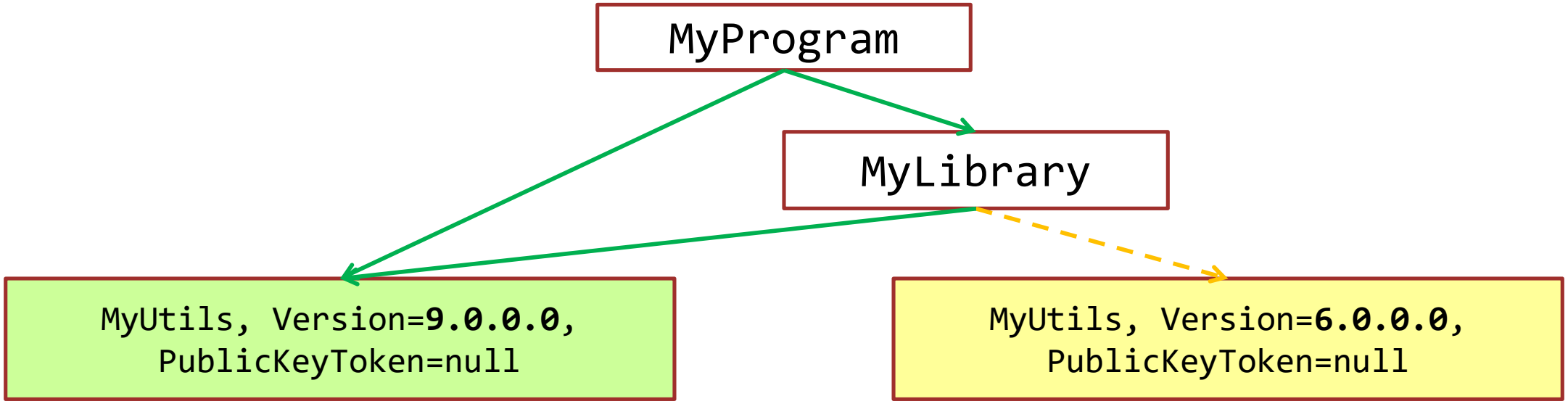
Strict assembly loading



Build stage: в BIN – `MyUtils.dll` максимальной версии (`9.0.0.0`)

?

Strict assembly loading

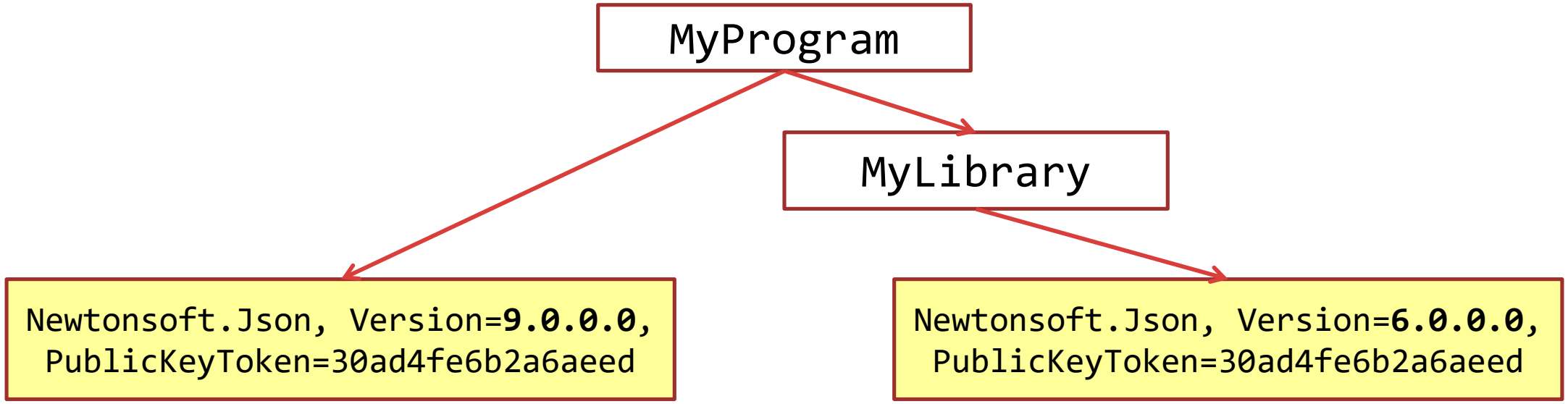


Build stage: в BIN – `MyUtils.dll` максимальной версии (`9.0.0.0`)

Runtime:

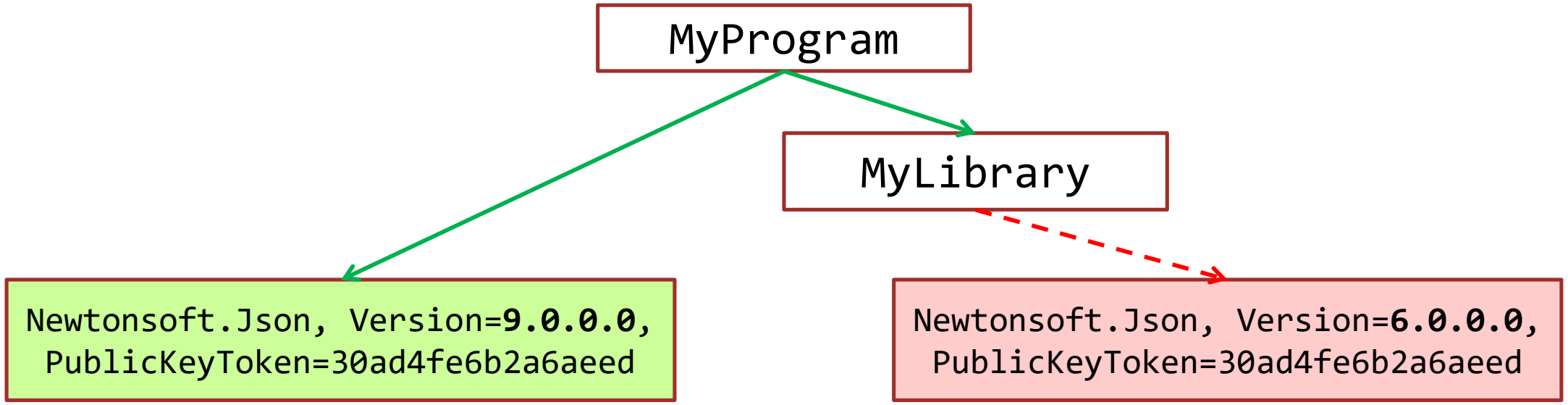
- **`MyProgram`:** `MyUtils.dll, Version=9.0.0.0`
- **`MyLibrary`:** `MyUtils.dll, Version=9.0.0.0`

Strict assembly loading



Build stage: в BIN – `Newtonsoft.Json.dll` максимальной версии (`9.0.0.0`)

Strict assembly loading



Build stage: в BIN – `Newtonsoft.Json.dll` максимальной версии (`9.0.0.0`)

Runtime:

- **`MyProgram:`** `MyUtils.dll, Version=9.0.0.0`
- **`MyLibrary:`** **`System.IO.FileLoadException`**

Binding redirect via App.config

```
<assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
   <dependentAssembly>
      <assemblyIdentity
         name="Newtonsoft.Json"
         publicKeyToken="30ad4fe6b2a6aeed"
         culture="neutral" />
      <bindingRedirect
         oldVersion="0.0.0.0-9.0.0.0"
         newVersion="9.0.0.0" />
   </dependentAssembly>
</assemblyBinding>
```

А ещё:
- publisher policy
- machine.config

https://docs.microsoft.com/en-us/dotnet/framework/configure-apps/redirect-assembly-versions

Упрощаем написание редиректов

- Никто не хочет писать Binding redirects вручную
- Можно отдать эту задачу MSBuild'у

Binding redirects: совет 1

Включить генерацию binding redirects средствами MsBuild при сборке проекта

```
// Add to *.csproj
<AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
```

- Результат: будут сгенерированы редиректы на версии сборок, находящиеся в BIN
- Работает только для проектов с OutputType Exe или WinExe

https://docs.microsoft.com/en-us/dotnet/framework/configure-apps/how-to-enable-and-disable-automatic-binding-redirection

Binding redirects: совет 2

Редиректы в тестах

Для старого формата *.csproj:

- Измените <OutputType> проекта с тестами тестами на Exe
- Или используйте хак:

```
<AutoGenerateBindingRedirects>true</AutoGenerateBindingRedirects>
<GenerateBindingRedirectsOutputType>true</GenerateBindingRedirectsOutputType>
```

Для нового формата *.csproj:

- Редиректы сгенерируются без вашего участия

Binding redirects: совет 3

- **Не используйте** генерацию редиректов средствами NuGet
- Редиректы придётся добавить в репозиторий (merge conflicts)
- Редиректы будут устаревать

Binding redirects: домашнее задание

Узнайте, как работает автогенерация редиректов

- grep "AutogenerateBindingRedirects" in dotnet folder
  - sdk/3.0.100/Microsoft.Common.CurrentVersion.targets
  - sdk/3.0.100/Sdks/Microsoft.NET.Sdk/targets/Microsoft.NET.Sdk.BeforeCommon.targets
- ResolveAssemblyReferences Task
  - Generates <SuggestedRedirects> items
- GenerateBindingRedirects Task
  - Write ResolveAssemblyReference output to .exe.config file

https://github.com/microsoft/msbuild/blob/master/src/Tasks/AssemblyDependency/ResolveAssemblyReference.cs
https://github.com/microsoft/msbuild/blob/master/src/Tasks/AssemblyDependency/GenerateBindingRedirects.cs

Альтернатива XML-конфигам

- Приложению требуется загрузить плагин
- Сборка плагина имеет зависимости и требует редиректов
- Способы:
  - Создать новый AppDomain, указав *.config файлов
  - Обрабатывать ошибки загрузки сборок в рантайме

Альтернатива XML-конфигам

```
AppDomain.CurrentDomain.AssemblyResolve += (sender, eventArgs) =>
{
    var name = eventArgs.Name;
    var requestingAssembly = eventArgs.RequestingAssembly;

    return Assembly.LoadFrom(...); // PublicKeyToken should be equal
};
```

AppDomainManager

- `AppDomain.CurrentDomain.AssemblyResolve`
- Что делать, если плагин создаёт AppDomain внутри себя?

```
<runtime>
  <appDomainManagerType value="..." />
  <appDomainManagerAssembly value="..." />
</runtime>
```

Strict assembly loading & .NET core

- В .NET Core такой проблемы нет!
- Но для всех сборок загружается только версия ≥ требуемой

.NET Core redirect

```
AppDomain.CurrentDomain.AssemblyResolve += (s, eventArgs) =>
{
    CheckForRecursion();

    var name = eventArgs.Name;
    var requestingAssembly = eventArgs.RequestingAssembly;

    name.Version = new Version(0, 0);

    return Assembly.Load(name);
};
```

.NET Core redirect

- Надо загрузить: `MyUtils, Version=0.0.2.0`
- Находится в BIN: `MyUtils, Version=0.0.1.0`
- `Redirect: 0.0.2.0 -> 0.0`

```
System.IO.FileNotFoundException:
Could not load file or assembly
'MyUtils, Version=0.0.65535.65535, ...'
```

.NET Core redirect

```
new Version(0, 0) == new Version(0, 0, -1, -1)

class Version {
  readonly int _Build;
  readonly int _Revision;
  readonly int _Major;
  readonly int _Minor;
}


(ushort) -1 == 65535
```

Assembly introspection

Получить все типы из сборки:

```
Type[] types = assembly.GetTypes();
```

Проблема:

- Не все типы могут загрузиться
- `class Clazz : TypeFromBadAssembly {}`

Assembly introspection

```
static IEnumerable<Type> GetTypesSafe(this Assembly assembly)
{
    try
    {
        return assembly.GetTypes();
    }
    catch (ReflectionTypeLoadException e)
    {
        return e.Types.Where(x => x != null);
    }
}
```

2. Strong naming

Strict Assembly loading относится только к Strong Named сборкам

Что такое Strong Name?

- Strong naming – подпись сборки приватным ключом (*.snk)

Зачем его используют?

- Позволяет различать сборки с одинаковыми именами
- Установка в GAC, загрузка нескольких версий side-by-side, …
- Strong named сборки могут ссылаться только на strong named сборки

https://docs.microsoft.com/en-us/dotnet/standard/assembly/strong-named

Strong Name: легаси?

"Do not rely on strong names for security.
They provide a unique identity only." – MSDN

- Ключ сборки нельзя изменить – это сломает binding redirects
- И даже если приватный ключ утёк. Механизм отзыва ключа не предусмотрен

https://docs.microsoft.com/en-us/dotnet/standard/library-guidance/strong-naming

Strong name: легаси?

Рекомендации из гайда по Open-source библиотекам:

- Дополнительно использовать Authenticode, NuGet package signing
- Коммитить приватный ключ в репозиторий, чтобы было проще форкать
- Никогда не менять Strong Name Key, чтобы не наносить лишний урон

https://docs.microsoft.com/en-us/dotnet/standard/library-guidance/strong-naming

Strong name change: example

Rx-Linq/2.3.0

```
System.Reactive.Linq, Version=2.3.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35 // Microsoft
```

System.Reactive.Linq/3.0.0

```
System.Reactive.Linq, Version=3.0.0.0,
Culture=neutral, PublicKeyToken=94bc3704cddfc263 // .NET Foundation
```

Как сделать редирект?

НИКАК

Strong name validation

Ещё один аргумет, почему Strong Naming – не о безопасности

Задача:

- Исправить баг в подписанной сборке без доступа к исходникам

Решение:

- dnSpy

Последствия:

- Всё работало (Strong name validation bypass – включен по умолчанию)
- Но упало на IIS'е

https://docs.microsoft.com/en-us/dotnet/standard/assembly/disable-strong-name-bypass-feature

Versioning policy

- Цель, как разработчика библиотеки: уменьшить количество редиректов у пользователя
- Не менять Assembly Version постоянно
  - При установке в GAC может загрузиться не та версия, которая ожидается
  - Выбрано как официальная рекомендация, как меньшее зло

Пример: `Newtonsoft.Json`

- NuGet versions: `12.0.1, 12.0.2, 12.0.3-beta*`
- Assembly Version: `12.0.0.0`

https://blogs.msdn.microsoft.com/suzcook/2003/05/30/when-to-change-fileassembly-versions/
https://docs.microsoft.com/en-us/dotnet/standard/library-guidance/strong-naming

Выводы

- Следуйте советам для .NET Framework
  - Включите автогенерацию редиректов
  - Старайтесь использовать одинаковые версии зависимостей во всех проектах в solution
- Strong naming нужен только если этого требует сценарий загрузки сборок, либо вы разрабатываете библиотеку
- Не меняйте Strong Name Key в своих библиотеках

3. .NET Standard

- Средство для написания библиотек, совместимых с различными реализациями .NET
- Реализации - .NET Framework, .NET Core, Mono, Unity, Xamarin

.NET Standard: нужен ли в 2020?

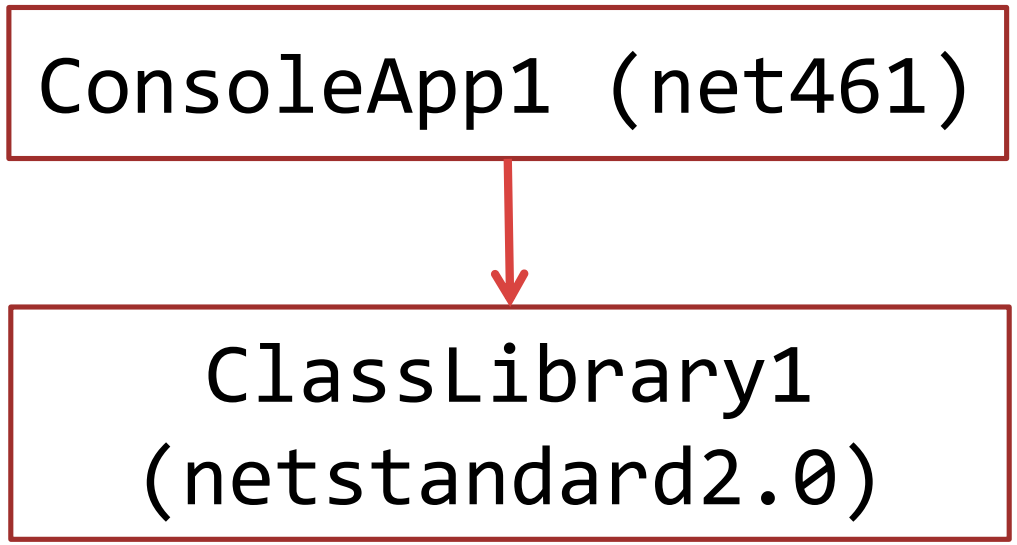
.NET Standard	1.4	2.0	2.1
.NET Core	1.0	2.0	3.0
.NET Framework	4.6.1	4.6.1*	N/A
Mono	4.6	5.4	6.4

- .NET Framework 4.8 поддерживает только .NET Standard 2.0
- Релиза с поддержкой поздних версий – не запланировано
- Библиотеки «заблокированы» на .NET Standard 2.0

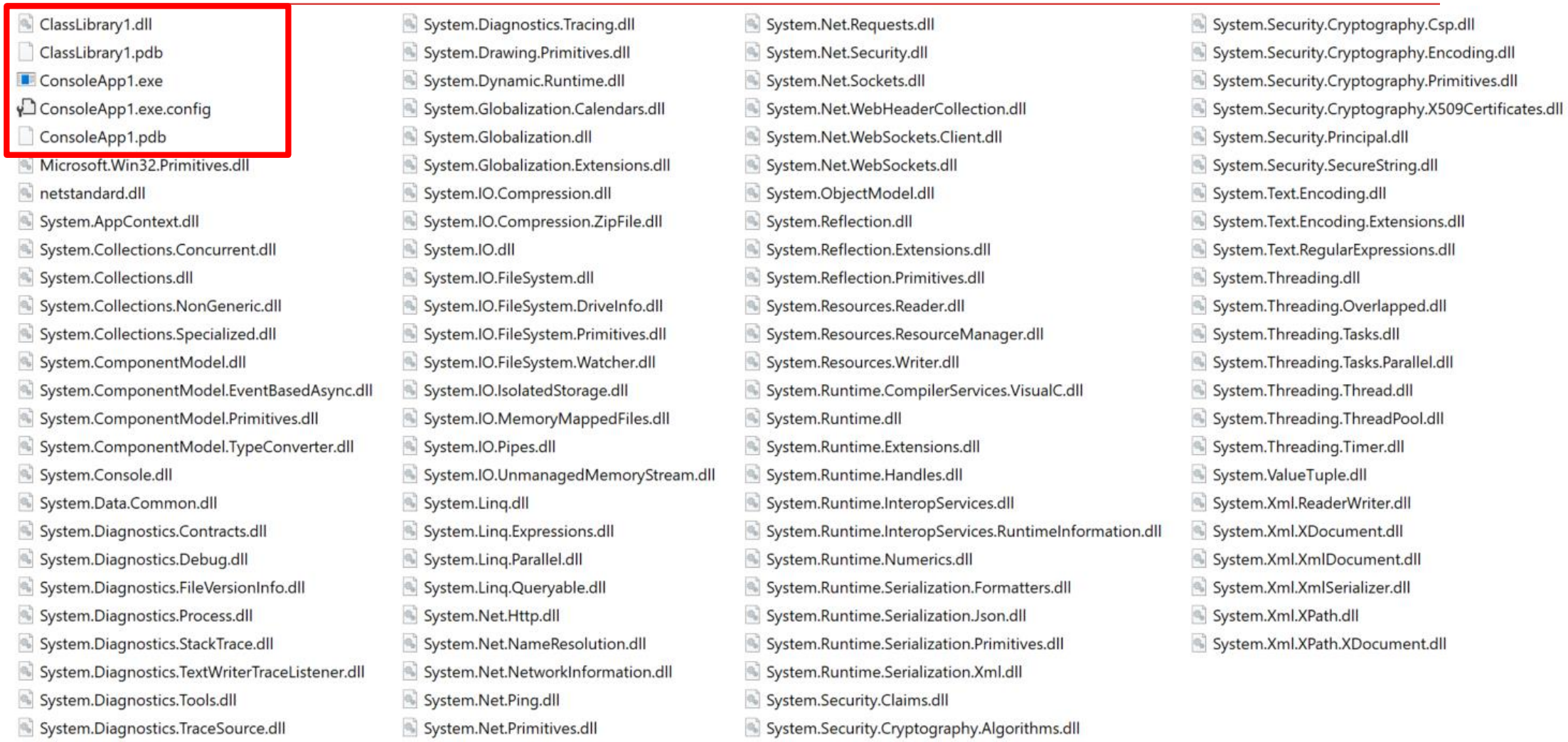
* https://docs.microsoft.com/en-us/dotnet/standard/net-standard

.NET Standard 2.0 и .NET Framework

- Приложение на .NET Framework с одной .NET Standard зависимостью



BIN получившегося приложения



.NET Standard 2.0 и .NET Framework

Target Framework	Extra DLL count
4.6.1	96
4.6.2	96
4.7	96
4.7.1	12
4.7.2	0
4.8	0

Immo Landwerth
@terrajobst

Sorry but we messed up. We tried to make .NET Framework 4.6.1 retroactively implement .NET Standard 2.0. This was a mistake as we don't have a time machine and there is a tail of bugs.

If you want to consume .NET Standard 1.5+ from .NET Framework, I recommend to be on 4.7.2.

> 
> **Marc Gravell** @marcgravell · Aug 21, 2018
> Sigh; another report of problems with System.Numerics.Vectors in a .net standard 2.0 lib running on net461 causing binding problems. Yay?

.NET Standard: выводы

- По возможности поднимите Target Framework до 4.7.1
- Делайте в библиотеках отдельный таргет для .NET Framework – упростите жизнь пользователям

```
<TargetFrameworks>netstandard20;net461</TargetFrameworks>
```

4. .NET Core

- Общая теория
- Запуск JetBrains Rider
  - Особенности solution
  - Хаки для загрузки сборок

.NET Core приложение

Сценарии:

- Framework-dependent deployment
- Self-contained
- Single-file (just self-extracting archive)

.NET Core приложение: BIN

- `MyProgram.dll`
- `MyProgram.runtimeconfig.json`
- `MyProgram.deps.json`
- `MyProgram.exe`
- `Dependencies`

.NET Core: .runtimeconfig.json

- Настройки рантайма, задающиеся при старте
- Shared Framework

```
{
  "runtimeOptions": {
    "framework": {
      "name": "Microsoft.NETCore.App",
      "version": "3.0.0"
    }
  }
}
```

.NET Core: runtimeconfig.template.json

- .runtimeconfig.json генерируется при сборке проекта
- Как добавить в него свои настройки:
  - *.csproj property (если предусмотрено)

```
<ServerGarbageCollection>true</ServerGarbageCollection>
```

  - runtimeconfig.template.json

```
{
  “configProperties": {
    "System.GC.Server": true
  }
}
```

Shared framework

- Shared framework – рантайм и набор библиотек
- Можно установить несколько версий
- Могут наследоваться

Хранятся в:

- `C:\Program Files\dotnet\shared\{name}\{version}`
- `/usr/bin/share/dotnet/shared/{name}/{version}`

Shared framework: примеры

- `Microsoft.NETCore.App`
- `Microsoft.AspNetCore.App`
- `Microsoft.WindowsDesktop.App`

Dependency manifest: .deps.json

- Файл с описанием всех зависимостей сборки/shared FW
- Если .deps.json нету – приложение может использовать сборки из рантайма и своей BIN директории
- Если .deps.json есть:
  - Если нет сборки, указанной в .deps.json – приложение не запустится
    - An assembly specified in the application dependencies manifest was not found
  - Рантайм будет игнорировать сборки, которых нет в .deps.json
    - `System.IO.FileNotFoundException: Could not load file or assembly`

https://docs.microsoft.com/en-us/dotnet/core/dependency-loading/default-probing

JetBrains Rider

Rider – .NET IDE

Backend Rider – кроссплатформенное .NET приложение

Где запускается?

- Windows: .NET Framework
- Linux & OSX: Mono

JetBrains Rider: ultimate goal

Задача: запустить на .NET Core, чтобы:

- Улучшить производительность
- Уменьшить потребление памяти
- Отказаться от legacy
- Контролировать версию рантайма
- И использовать новые API

JetBrains Rider: current state

Задача: запустить прототип на .NET Core, чтобы:

- Снизить технологические риски

JetBrains Rider on .NET Core

Особенности:

- Visual Studio (даже без R#) падает с OOM на больших Solution, если в них есть проекты в SDK-style `*.csproj`
- Разработчики R# используют Visual Studio
- В R# есть ссылки на Framework-специфичные библиотеки
  - Windows: `Microsoft.WindowsDesktop.App`, Compatibility Pack
  - Linux & OSX: Mock-сборки с минимальной функциональностью

JetBrains Rider on .NET Core

Решение:

- Остаться на старых `*.csproj` и собирать под полный Framework
- Добавить в проект самописный `.runtimeconfig.json`
- Загружать .NET Core версии зависимостей, если таковые имеются

JetBrains Rider: .NET Core trick #1

Задача:

- Вызвать метод, который есть только в .NET Framework

Проблема:

- Вызывающий метод не скомпилируется JIT'ом, будет MissingMethodException

JetBrains Rider: .NET Core trick #1

```
static void Method() {
  if (NetFramework)
      CallNETFrameworkOnlyMethod();

  ...
}

[MethodImpl(MethodImplOptions.NoInlining)]
static void CallNETFrameworkOnlyMethod() {
  NETFrameworkOnlyMethod();
}
```

JetBrains Rider: .NET Core trick #2

Задача:

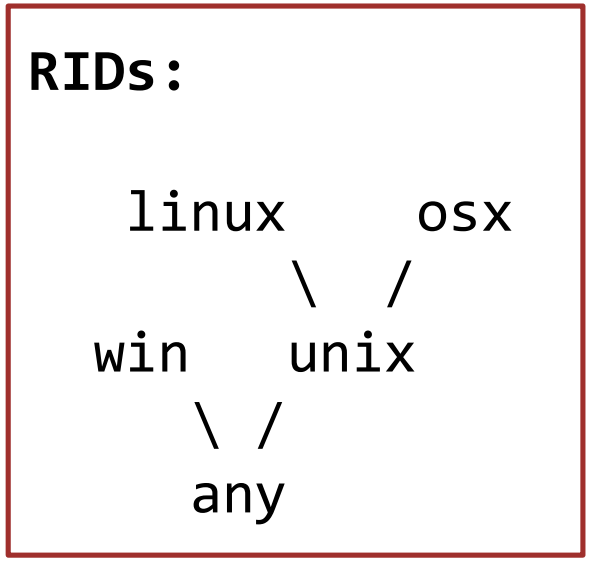
- Загрузить сборку по относительному пути

- `NuGet.Common.dll – for .NET Framework & Mono`
- `NetCore/NuGet.Common.dll – for .NET Core`

JetBrains Rider: .NET Core trick #2

```
"System.Diagnostics.PerformanceCounter/4.6.0": {
  "runtime": {
    "lib/.../System.Diagnostics.PerformanceCounter.dll": {
      ...
    }
  },
  "runtimeTargets": {
    "runtimes/win/.../System.Diagnostics.PerformanceCounter.dll": {
      "rid": "win",
      ...
    }
  }
}
```

RIDs:



JetBrains Rider: .NET Core trick #3

Задача:

- Загрузить Mock для `WindowsBase.dll` на Unix-like системах
- Сборка с именем `WindowsBase` уже присутствует в `Microsoft.NETCore.App`

Решение:

- На Windows `WindowsBase.dll` из `Microsoft.WindowsDesktop.App` переопределяет generic-версию
- Посмотрим, как это реализовано

JetBrains Rider: .NET Core trick #3

- Microsoft.NETCore.App.deps.json

```
"runtimes/win-x64/lib/netcoreapp3.0/WindowsBase.dll": {
  "assemblyVersion": "4.0.0.0",
  "fileVersion": "4.700.19.46214"
}
```

- Microsoft.WindowsDesktop.App.deps.json

```
"runtimes/win-x64/lib/netcoreapp3.0/WindowsBase.dll": {
  "assemblyVersion": "4.0.0.0",
  "fileVersion": "4.800.19.46214"
}
```

JetBrains Rider: .NET Core trick #3

- Microsoft.NETCore.App.deps.json

```
"runtimes/win-x64/lib/netcoreapp3.0/WindowsBase.dll": {
  "assemblyVersion": "4.0.0.0",
  "fileVersion": "4.700.19.46214"
}
```

- Microsoft.WindowsDesktop.App.deps.json

```
"runtimes/win-x64/lib/netcoreapp3.0/WindowsBase.dll": {
  "assemblyVersion": "4.0.0.0",
  "fileVersion": "4.800.19.46214"
}
```

```
(4.0.0.0, 4.800.19.46214) > (4.0.0.0, 4.700.19.46214)
```

JetBrains Rider: .NET Core trick #4

Задача:

- .deps.json нужен только для отдельных зависимостей
- Хочется сохранить .deps.json минимальным
- Нужно разрешить загружать сборки из BIN, не указанные в .deps.json

JetBrains.ReSharper.Host.runtimeconfig.json

==========

```
{
  "configProperties": {
    "Microsoft.NETCore.DotNetHostPolicy.SetAppPaths": true
  }
}
```

Выводы

- .runtimeconfig.json + .deps.json – аналог App.config
- .deps.json позволяет кастомизировать загрузку сборок

5. Assembly loading debugging

- Логи загрузчика сборок
- Exceptions
- Runtime events

.NET Framework: Fusion logs

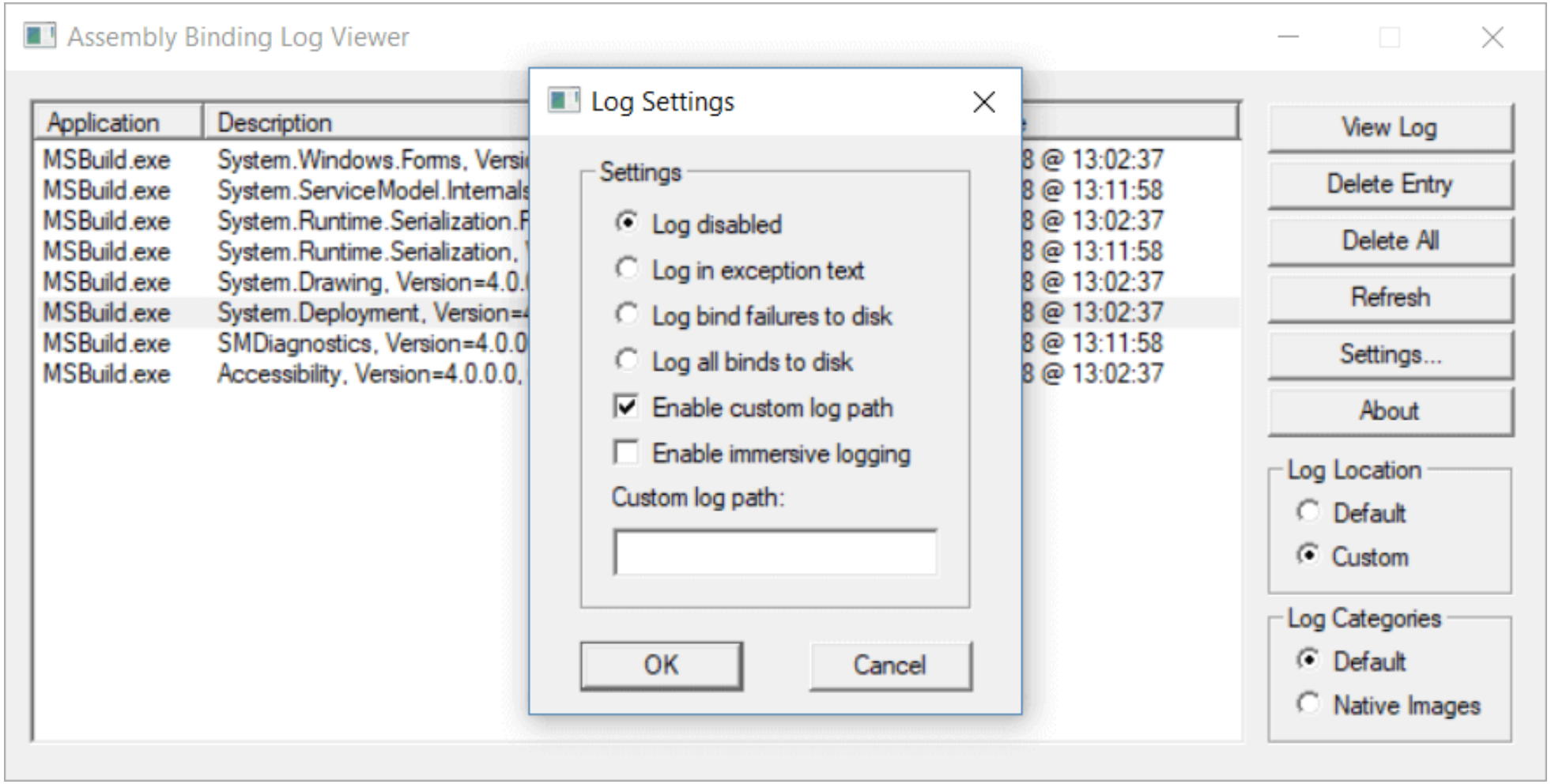
HKLM\Software\Microsoft\Fusion\ForceLog=1

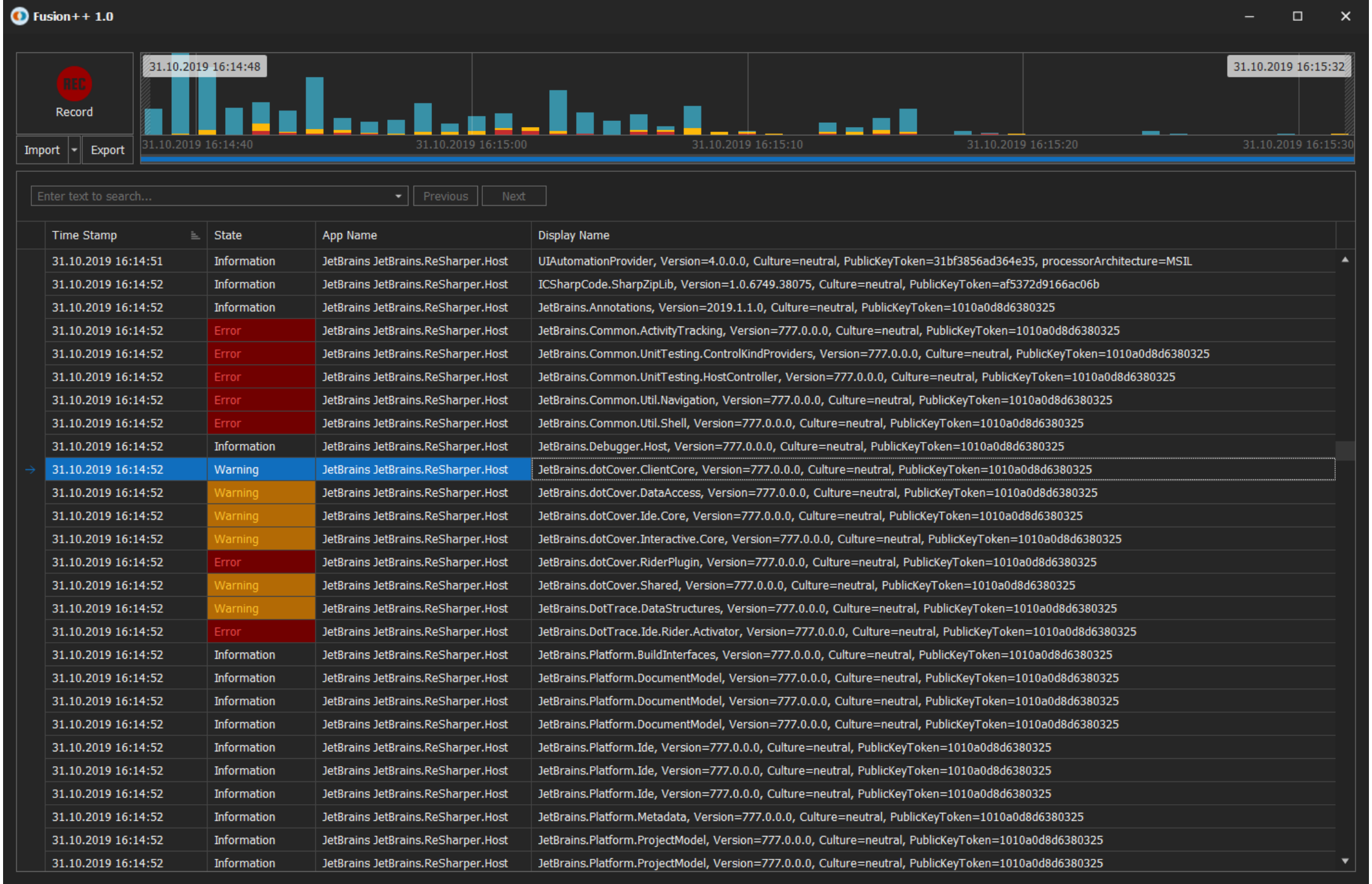
HKLM\Software\Microsoft\Fusion\LogPath=…

- fuslogvw (Fusion Log Viewer)
- Fusion++

https://www.hanselman.com/blog/BackToBasicsUsingFusionLogViewerToDebugObscureLoaderErrors.aspx
https://github.com/awaescher/Fusion

.NET Framework: fuslogvw




Fusion++ 1.0
Record
Import
Export
31.10.2019 16:14:48
31.10.2019 16:15:32
31.10.2019 16:14:40
31.10.2019 16:15:00
31.10.2019 16:15:10
31.10.2019 16:15:20
31.10.2019 16:15:30
Enter text to search...
Previous
Next
Time Stamp
State
App Name
Display Name
31.10.2019 16:14:51 Information JetBrains JetBrains.ReSharper.Host UIAutomationProvider, Version=4.0.0.0, Culture=neutral, PublicKeyToken=31bf3856ad364e35, processorArchitecture=MSIL
31.10.2019 16:14:52 Information JetBrains JetBrains.ReSharper.Host ICSharpCode.SharpZipLib, Version=1.0.6749.38075, Culture=neutral, PublicKeyToken=af5372d9166ac06b
31.10.2019 16:14:52 Information JetBrains JetBrains.ReSharper.Host JetBrains.Annotations, Version=2019.1.1.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Error JetBrains JetBrains.ReSharper.Host JetBrains.Common.ActivityTracking, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Error JetBrains JetBrains.ReSharper.Host JetBrains.Common.UnitTesting.ControlKindProviders, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Error JetBrains JetBrains.ReSharper.Host JetBrains.Common.UnitTesting.HostController, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Error JetBrains JetBrains.ReSharper.Host JetBrains.Common.Util.Navigation, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Error JetBrains JetBrains.ReSharper.Host JetBrains.Common.Util.Shell, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Information JetBrains JetBrains.ReSharper.Host JetBrains.Debugger.Host, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Warning JetBrains JetBrains.ReSharper.Host JetBrains.dotCover.ClientCore, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Warning JetBrains JetBrains.ReSharper.Host JetBrains.dotCover.DataAccess, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Warning JetBrains JetBrains.ReSharper.Host JetBrains.dotCover.Ide.Core, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Warning JetBrains JetBrains.ReSharper.Host JetBrains.dotCover.Interactive.Core, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Error JetBrains JetBrains.ReSharper.Host JetBrains.dotCover.RiderPlugin, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Warning JetBrains JetBrains.ReSharper.Host JetBrains.dotCover.Shared, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Warning JetBrains JetBrains.ReSharper.Host JetBrains.DotTrace.DataStructures, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Error JetBrains JetBrains.ReSharper.Host JetBrains.DotTrace.Ide.Rider.Activator, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Information JetBrains JetBrains.ReSharper.Host JetBrains.Platform.BuildInterfaces, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Information JetBrains JetBrains.ReSharper.Host JetBrains.Platform.DocumentModel, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Information JetBrains JetBrains.ReSharper.Host JetBrains.Platform.DocumentModel, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Information JetBrains JetBrains.ReSharper.Host JetBrains.Platform.DocumentModel, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Information JetBrains JetBrains.ReSharper.Host JetBrains.Platform.Ide, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Information JetBrains JetBrains.ReSharper.Host JetBrains.Platform.Ide, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Information JetBrains JetBrains.ReSharper.Host JetBrains.Platform.Ide, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Information JetBrains JetBrains.ReSharper.Host JetBrains.Platform.Metadata, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Information JetBrains JetBrains.ReSharper.Host JetBrains.Platform.ProjectModel, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325
31.10.2019 16:14:52 Information JetBrains JetBrains.ReSharper.Host JetBrains.Platform.ProjectModel, Version=777.0.0.0, Culture=neutral, PublicKeyToken=1010a0d8d6380325

Fusion logs: example

```
Unhandled Exception: System.IO.FileLoadException: Could not load
file or assembly 'Newtonsoft.Json, Version=9.0.0.0,
Culture=neutral, PublicKeyToken=30ad4fe6b2a6aeed' or one of its

LOG: Post-policy reference: Newtonsoft.Json, Version=9.0.0.0,
Culture=neutral, PublicKeyToken=30ad4fe6b2a6aeed

LOG: Assembly download was successful. Attempting setup of file:
BIN\Newtonsoft.Json.dll

LOG: Assembly Name is: Newtonsoft.Json, Version=6.0.0.0,
Culture=neutral, PublicKeyToken=30ad4fe6b2a6aeed

WRN: Comparing the assembly name resulted in the mismatch: Major
Version

ERR: The assembly reference did not match the assembly definition
found.
```

Mono logs

```
export MONO_LOG_MASK=asm
export MONO_LOG_LEVEL=debug
```

Логи пишутся в `stdout/stderr`

https://www.mono-project.com/docs/advanced/runtime/logging-runtime-events/

.NET Core logs

```
COREHOST_TRACE=1
```

```
Output:
```

- .NET Core 2.1: stderr
- .NET Core 3.0: COREHOST_TRACEFILE

https://docs.microsoft.com/en-us/dotnet/core/dependency-loading/default-probing
https://github.com/dotnet/core-setup/blob/master/Documentation/design-docs/host-tracing.md

Exception events

```
AppDomain.CurrentDomain.AssemblyResolve
AppDomain.CurrentDomain.FirstChanceException
```

ETW Exceptions (perfview)

Выводы

- Перекладывайте работу на средства разработки
- Переходите на .NET Core, чтобы забыть о Strict Assembly Loading и использовать новые API
- Используйте последние версии .NET Framework с полной поддержкой .NET Standard 2.0
- Применяйте готовые хаки и придумывайте свои в «безвыходных» ситуациях
- Вооружайтесь средствами отладки

Links

- .NET Guide

  https://docs.microsoft.com/en-us/dotnet/standard/
- Deep dive into .NET Core primitives (1, 2, 3)

  https://natemcmaster.com/blog/2017/12/21/netcore-primitives/

Вопросы

Евгений Пешков

telegram/twitter: @epeshk