Rainer Stropek | time cockpit

C#-Revolution

Your Host

Rainer Stropek

- Developer, Entrepreneur
- Azure MVP, MS Regional Director
- Trainer at IT-Visions

Contact

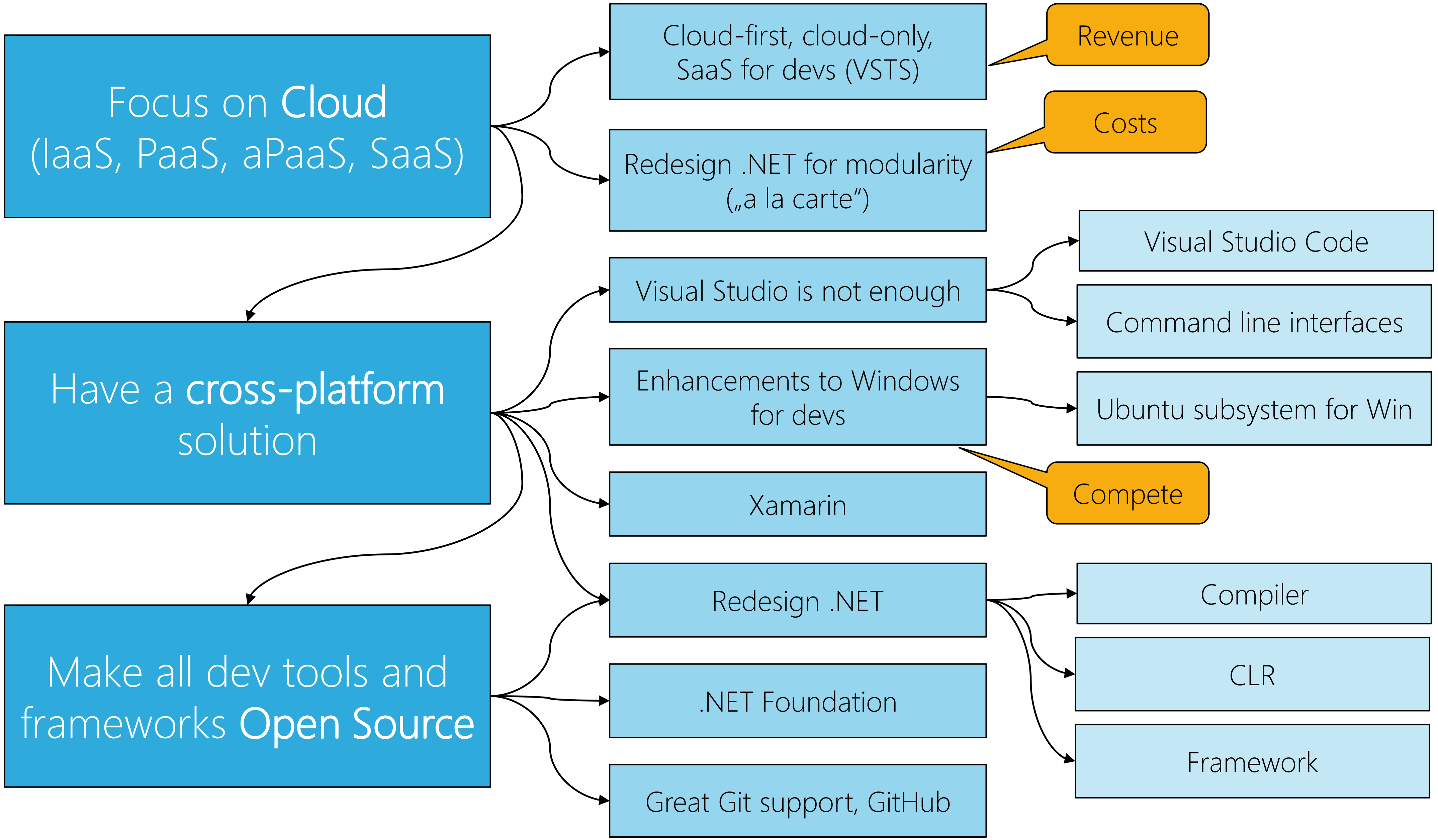
- software architects gmbh
  rainer@timecockpit.com
  Twitter: @rstropek

Agenda

C# und .NET machen einen radikalen Wandel durch. Open Source, Plattformunabhängigkeit, grundlegendes Redesign, neue Compilerplattform – als C#-Entwicklerinnen und -Entwickler gibt es viel Neues zu lernen. Der BASTA!-C#-Workshop von Rainer Stropek ist eine gute Gelegenheit, sich einen Tag Zeit zu nehmen, um auf den neuesten Stand zu kommen. Im Workshop werden unter anderem folgende Themen behandelt:

- Neuerungen in C# und Visual Studio
- Die neue .NET Runtime
- dotnet CLI
- Die neue .NET-Ausführungsumgebung
- Anwendungsbeispiele in ASP.NET Core 1 (Fokus liegt auf der Sprache und .NET-Grundlagen, nicht auf ASP.NET)
- Neue Tools und Libraries.

In der bewährten Art und Weise wird sich Rainer Stropek im Workshop auf Codebeispiele statt Slides konzentrieren.


Focus on Cloud (IaaS, PaaS, aPaaS, SaaS)
Cloud-first, cloud-only, SaaS for devs (VSTS)
Revenue
Redesign .NET for modularity („a la carte“)
Costs
Have a cross-platform solution
Visual Studio is not enough
Visual Studio Code
Command line interfaces
Enhancements to Windows for devs
Ubuntu subsystem for Win
Compete
Xamarin
Make all dev tools and frameworks Open Source
Redesign .NET
Compiler
CLR
Framework
.NET Foundation
Great Git support, GitHub

.NET Core

Why .NET Core?

Refactor .NET Framework

- Establish a Standard Library for the various incarnations of .NET
- .NET Core is not 100% compatible with .NET 4.x (details)

Make it a real cross-platform solution

- Windows, Mac OS, Linux (details)

Make it open source

- A .NET Foundation project
- MIT License

Details: https://docs.microsoft.com/dotnet/

Components of .NET Core

.NET Runtime (CoreCLR)

CoreCLR includes Base Class Library (BCL)

.NET Core Foundation Libraries (CoreFX)

.NET Command Line Tools (.NET CLI)

Including the **`dotnet`** application host

Cross-Platform Compiler (Roslyn)

Status of .NET Core

.NET Core 1.0 is RTM

- 1.0.1 published recently (details), 1.1 is scheduled for Fall 2016

Visual Studio Tools are in preview (download)

C# is RTM

- VB and F# are coming

X64 Support

- X86 support on Windows
- ARM support will come

See also: https://github.com/dotnet/core/blob/master/roadmap.md

What can you build?

Console applications

ASP.NET Core applications

UWP applications

Xamarin Forms applications

See also: https://github.com/dotnet/core/blob/master/roadmap.md

Where to get .NET Core?

.NET Core landing page
- With Visual Studio tools (Visual Studio prerequisites)
- Command-line tools (with your own editor, e.g. VSCode)

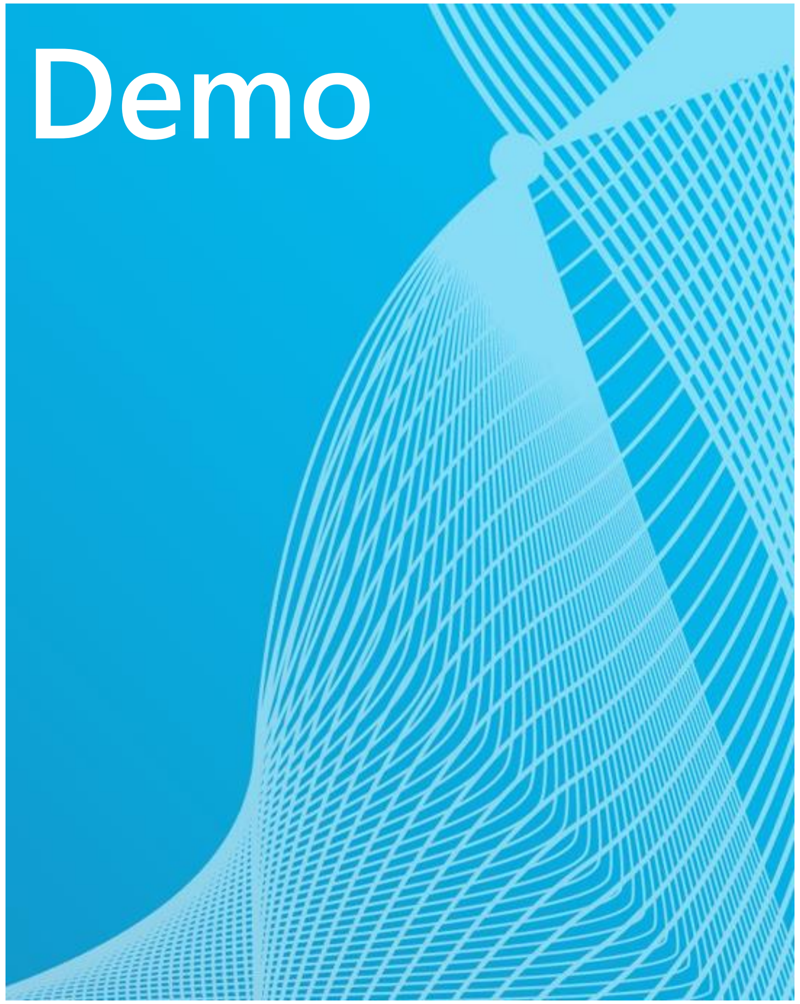
.NET Install Script (details, download)
- You have to care for the prerequisites

NuGet
- Packages and Metapackages

Docker: `microsoft/dotnet` image (details)

See also: https://github.com/dotnet/core/blob/master/roadmap.md

Packages, Metapackages and Frameworks



Create console app with CLI

Analyze **`project.json`**

Discuss **`project.json`** reference

Run app

Further readings

- [More about cross-platform libraries]()
- [Changes to project.json]()

https://github.com/rstropek/Samples/tree/master/AspNetCore1Workshop/10-console-hello-world

Console App

```
{
  "version": "1.0.0-*",
  "buildOptions": {
    "debugType": "portable",
    "emitEntryPoint": true
  },
  "dependencies": {},
  "frameworks": {
    "netcoreapp1.0": {
      "dependencies": {
        "Microsoft.NETCore.App": {
          "type": "platform",
          "version": "1.0.1"
        }
      },
      "imports": "dnxcore50"
    }
  }
}
```

Portable PDB (would be „full“ for Windows-only)

„true“ for executable, „false“ for libraries

Target Framework (details)

Metapackage for .NET Core distribution (details, NuGet)

Dependency is part of platform, not local deployment (details)

For migrating DNX-based apps only (details)

Project.json Reference: https://docs.microsoft.com/en-us/dotnet/articles/core/tools/project-json

Library

```
{
  "version": "1.0.0-*",
  "buildOptions": {
    "debugType": "portable"
  },
  "dependencies": {},
  "frameworks": {
    "netstandard1.6": {
      "dependencies": {
        "NETStandard.Library":
          "1.6.0"
      }
    }
  }
}
```

Portable PDB (would be „full“ for Windows-only)

Target Framework (details)

Metapackage for .NET Standard Library (details, NuGet)

Project.json Reference: https://docs.microsoft.com/en-us/dotnet/articles/core/tools/project-json

Highlights in `project.json`

`testRunner` – Test runner (e.g. `xUnit`, `mstest`; details later)

`shared` – shared files for library export (details later)

`dependencies` – framework-independent dependencies

`tools` – tools for build and deployment process (details later)

`scripts` – Script to run during build process (e.g. web dev tools)

`buildOptions` – Compiler options (can be framework-specific)

`publishOptions` – include/exclude patterns for build/publish

`runtimeOptions` – parameters for .NET runtime (e.g. GC)

https://docs.microsoft.com/en-us/dotnet/articles/core/tools/project-json

Cross-platform

Demo

Run app on Linux using Docker

.NET CLI

.NET Core CLI

`dotnet` command

- `new` – create project
- `restore` – restore dependencies
- `run` – run source code without explicit compile
- `build` – builds project and dependencies
- `test` – runs unit tests
- `pack` – packs code into a NuGet package
- `publish` – packs the app and dependencies for publishing

https://docs.microsoft.com/en-us/dotnet/articles/core/tools/dotnet

`dotnet run`

Run application from the source code

- Use **`dotnet`** without any command to run a built DLL

Uses **`dotnet build`** in the background

https://docs.microsoft.com/en-us/dotnet/articles/core/tools/dotnet

Deployment (`dotnet publish`)

Framework-dependent deployment

- Shared system-wide version of .NET Core must be present on target system
- DLLs are launched using **`dotnet`**
- DLLs are portable

Self-contained deployment

- No prerequisites on target system necessary
- Does *not* contain native prerequisites
- Results in an platform-specific executable

Optional: Use CrossGen for native image generation

Self-contained Deployment

Demo

Change **`project.json`** for SCD

- See following slides

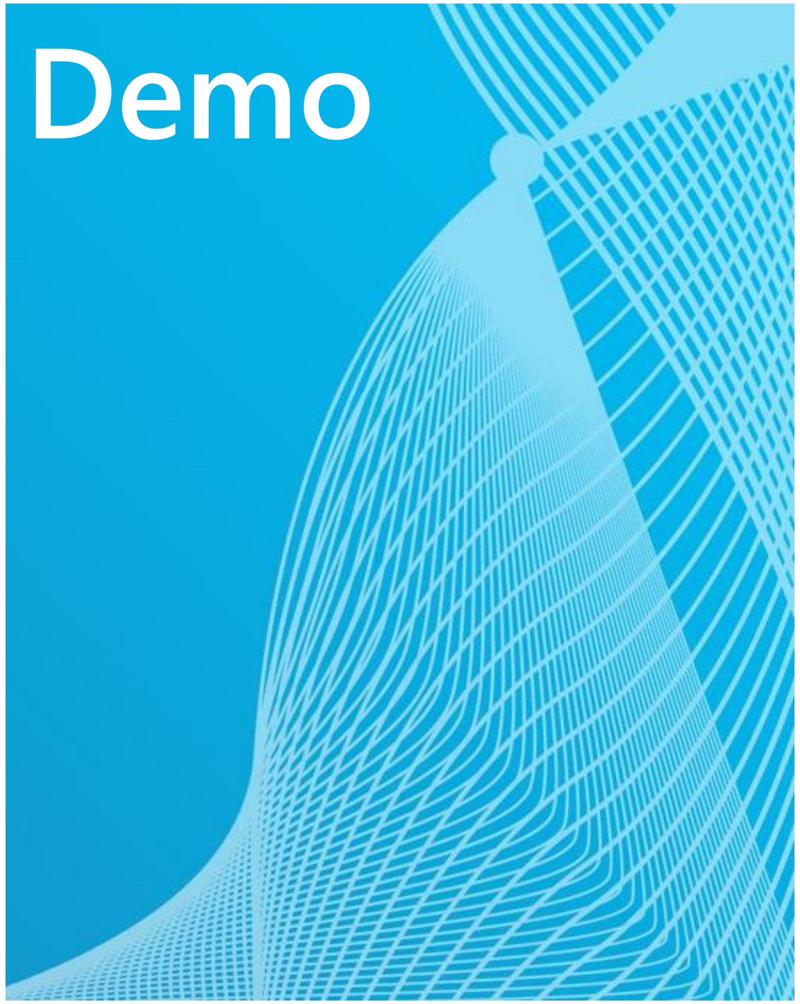
Build and publish SCD

```
dotnet restore
dotnet build -r win10-x64
dotnet publish -c release -r win10-x64
```

Runtime Identifier (RID) (details)

Release instead of debug version (need not ship PDBs)

Custom Tool



Create custom tool for dotnet CLI

- Create console app
- Update `project.json`

```
"outputName": "dotnet-classcount"
"dependencies": { "Microsoft.CodeAnalysis.CSharp": "1.3.2" }
```

Using custom tool

- Create library project

```
dotnet new -t Lib
```

- Add tool reference to `project.json`

```
"tools": { "ClassCounter": "1.0.0" }
```

- Restore and run

```
dotnet restore -f ..\ClassCounter\bin\Debug
dotnet classcount
```

https://github.com/rstropek/Samples/tree/master/AspNetCore1Workshop/45-custom-tool/

Self-contained Deployment

```
{
  "version": "1.0.0-*",
  "buildOptions": {
    "debugType": "portable",
    "emitEntryPoint": true
  },
  "dependencies": {
    "Microsoft.NETCore.App": "1.0.1",
    "Newtonsoft.Json": "9.0.1"
  },
  "frameworks": {
    "netcoreapp1.0": {}
  },
  "runtimes": {
    "win10-x64": {}
  }
}
```

Note: No type “platform” anymore

Details: https://docs.microsoft.com/en-us/dotnet/articles/core/deploying/index#self-contained-deployments-scd

Self-contained Deployment

```
{
  "version": "1.0.0-*",
  "buildOptions": {
    "debugType": "portable",
    "emitEntryPoint": true
  },
  "dependencies": {
    "NETStandard.Library": "1.6.0",
    "Microsoft.NETCore.Runtime.CoreCLR": "1.0.2",
    "Microsoft.NETCore.DotNetHostPolicy": "1.0.1",
    "Newtonsoft.Json": "9.0.1"
  },
  "frameworks": {
    "netstandard1.6": {}
  },
  "runtimes": {
    "win10-x64": {}
  }
}
```

Result: Approx. 30MB

Details: https://docs.microsoft.com/en-us/dotnet/articles/core/deploying/index#self-contained-deployments-scd

Versioning

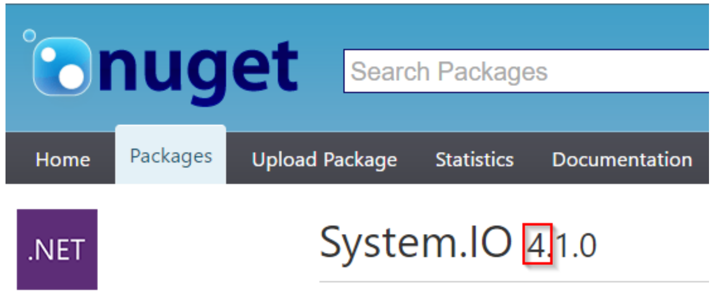
Versioning

Framework version changes when APIs are added

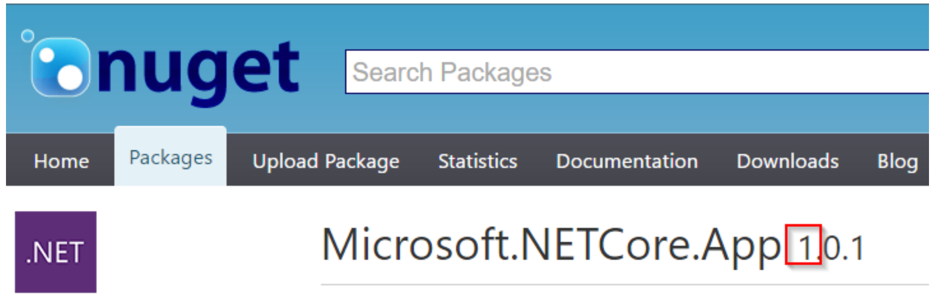
- No implementation → no patch numbers
- Example: `netcoreapp1.0`

Package versions

- `System.*` packages use 4.x numbers (overlap with .NET Framework)
- Packages without overlapping with .NET Framework → 1.x



System.IO 4.1.0



Microsoft.NETCore.App 1.0.1

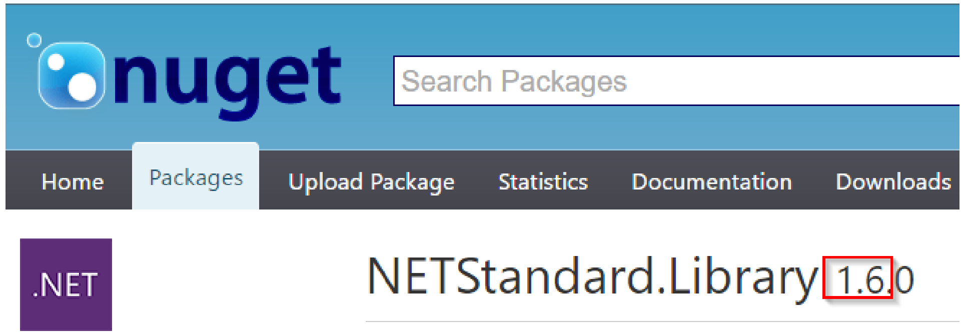
https://docs.microsoft.com/en-us/dotnet/articles/core/versions/index

Versioning

.NET Standard Library

Versioning independent of any .NET runtime, applicable to multiple runtimes

1.6 for .NET Core 1.0



NETStandard.Library 1.6.0

Examples

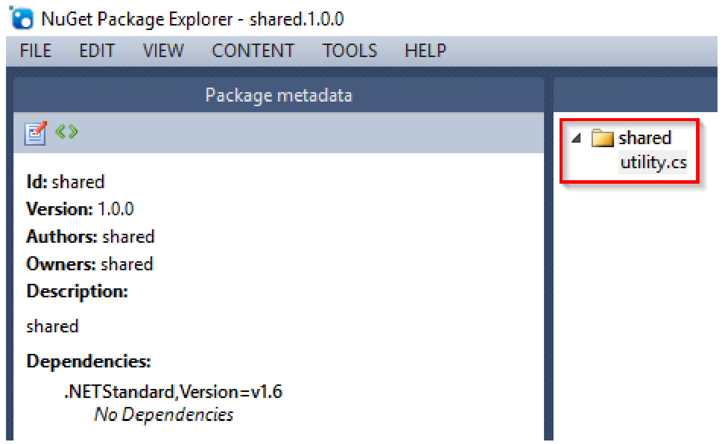
https://docs.microsoft.com/en-us/dotnet/articles/core/versions/index

Libraries

Sharing Files

Compile code in shared folder as if it was part of the project

Note: Use `internal` types only


NuGet Package Explorer - shared.1.0.0
FILE EDIT VIEW CONTENT TOOLS HELP
Package metadata

Id: shared
Version: 1.0.0
Authors: shared
Owners: shared
Description:

shared

Dependencies:

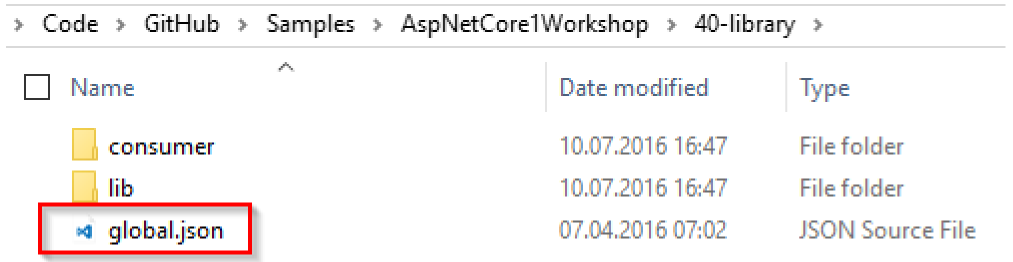
.NETStandard,Version=v1.6
No Dependencies

shared
utility.cs

Sample: https://github.com/rstropek/Samples/tree/master/AspNetCore1Workshop/30-shared-project

Libraries

Use `global.json` to specify folders

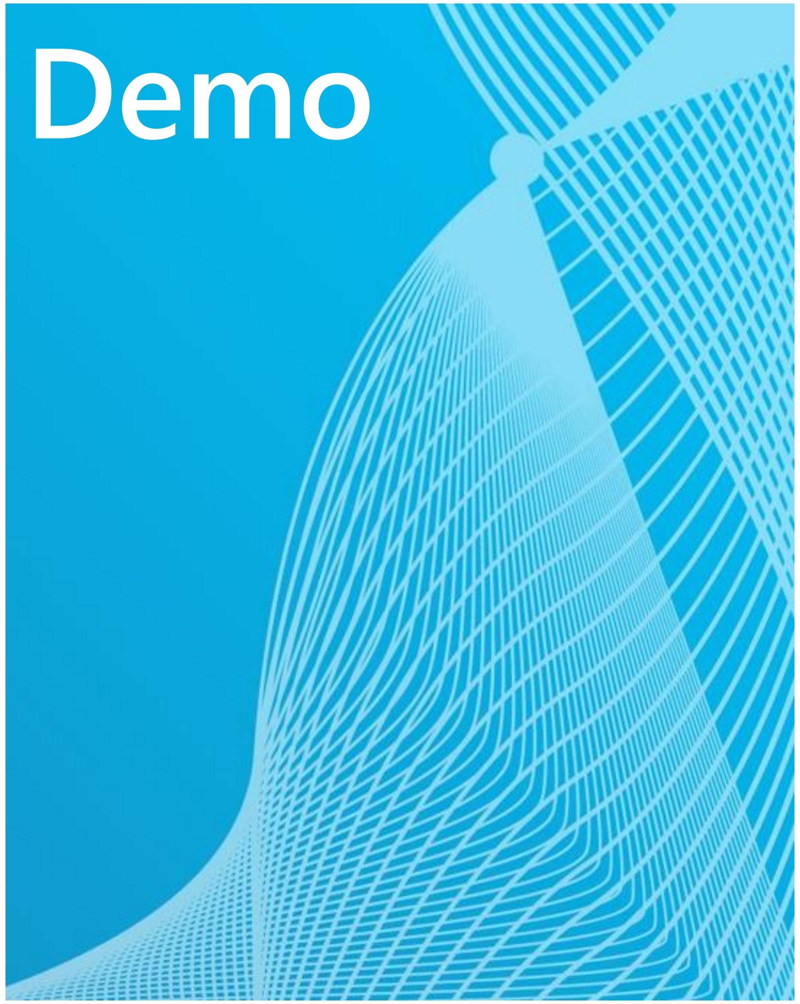


Code › GitHub › Samples › AspNetCore1Workshop › 40-library ›

Name	Date modified	Type
consumer	10.07.2016 16:47	File folder
lib	10.07.2016 16:47	File folder
global.json	07.04.2016 07:02	JSON Source File

Sample: https://github.com/rstropek/Samples/tree/master/AspNetCore1Workshop/40-library

Libraries



Shared files

Libraries

Creating NuGet packages

```
dotnet pack
```

Further readings

- More about cross-platform libraries
- Tools for porting code from .NET Framework

.NET Standard Library

Why a standard library?

CLR (CLI) has already been standardized (ECMA 334)

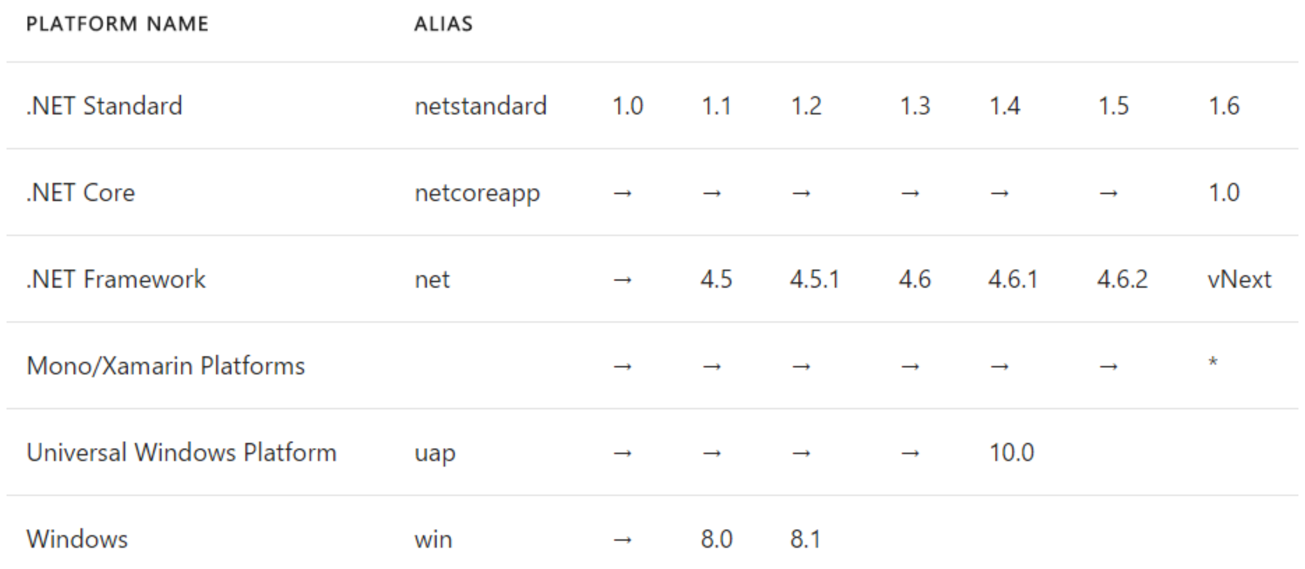
- No standardized BCL prior to .NET Core

Goal: Standard BCL API for all .NET platforms

- Easier to create portable libraries
- Reduce conditional compilation

What about PCLs?

- Well defined API instead of just intersection of platforms
- Better versioning
- Overlapping PCL profiles (details)



PLATFORM NAME	ALIAS							
.NET Standard	netstandard	1.0	1.1	1.2	1.3	1.4	1.5	1.6
.NET Core	netcoreapp	→	→	→	→	→	→	1.0
.NET Framework	net	→	4.5	4.5.1	4.6	4.6.1	4.6.2	vNext
Mono/Xamarin Platforms		→	→	→	→	→	→	*
Universal Windows Platform	uap	→	→	→	→	10.0		
Windows	win	→	8.0	8.1				

Details: https://docs.microsoft.com/en-us/dotnet/articles/standard/library

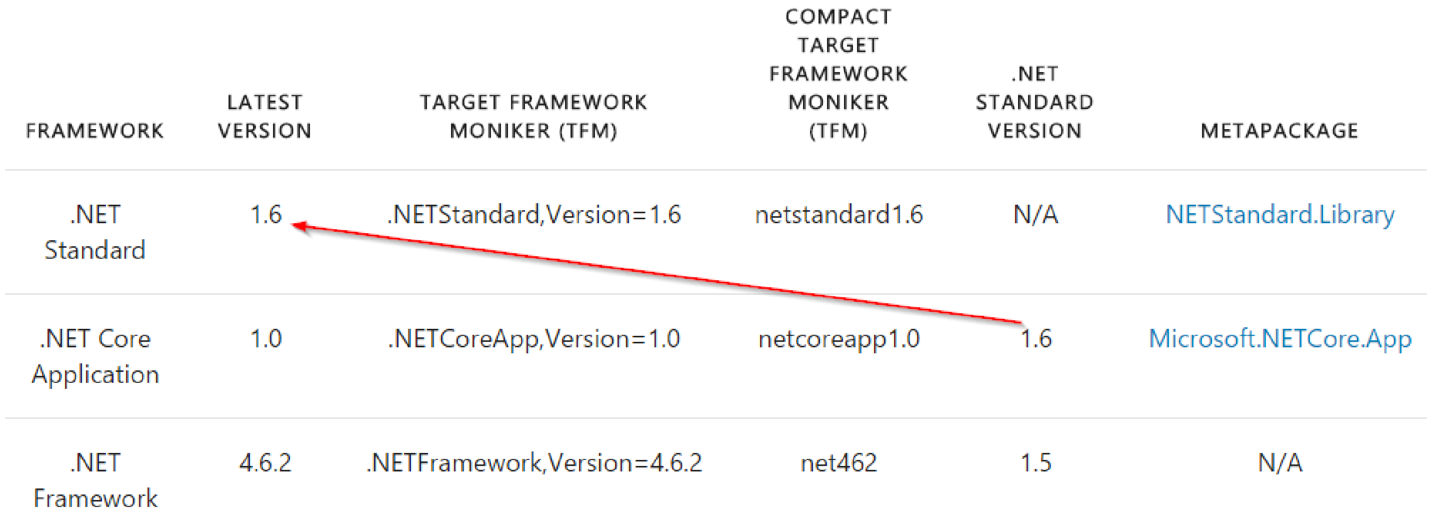
.NET Standard Library

Standard APIs defined as empty C# classes

Example: `ref` folder in `System.Runtime`

`NETStandard.Library` (NuGet)

Metapackage for .NET Standard Library



FRAMEWORK	LATEST VERSION	TARGET FRAMEWORK MONIKER (TFM)	COMPACT TARGET FRAMEWORK MONIKER (TFM)	.NET STANDARD VERSION	METAPACKAGE
.NET Standard	1.6	.NETStandard,Version=1.6	netstandard1.6	N/A	NETStandard.Library
.NET Core Application	1.0	.NETCoreApp,Version=1.0	netcoreapp1.0	1.6	Microsoft.NETCore.App
.NET Framework	4.6.2	.NETFramework,Version=4.6.2	net462	1.5	N/A

Details: https://docs.microsoft.com/en-us/dotnet/articles/standard/frameworks

ASP.NET Core 1 Basics

Practical use of .NET Core

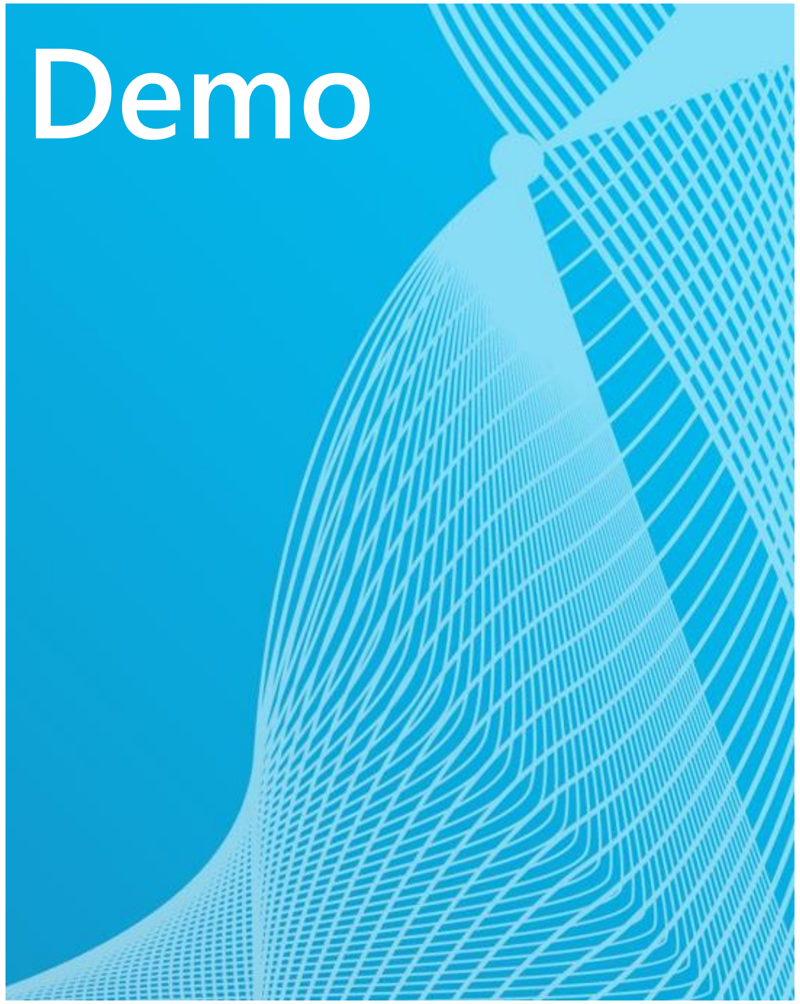
Minimal ASP.NET Core 1

Demo

- ASP.NET Pipeline
- Discuss “a la carte” framework
  - Add static files (sample)
- Kestrel
  - Windows, Linux with Docker
- Visual Studio Code
- Further readings
  - Building middlewares

https://github.com/rstropek/Samples/tree/master/AspNetCore1Workshop/50-simplest-aspnet

Walkthrough VS “File – New – Project”



Create web project in VS2015

Walkthrough

- Servers (IIS and Kestrel)
- Environments
- Adding MVC

101 for ASP.NET Core 1

[Application Startup]

- `Main` Method
- `Startup` class with [`ConfigureServices`] (DI) and `Configure` (Pipeline)

[Static Files]

[Environments]

[Servers]

- IIS, Kestrel

Configuration

No `web.config` anymore

Key/value pair settings from different providers

- E.g. memory, environment variables, JSON, INI, XML

Extensible

- Details about writing custom providers

Options pattern for DI integration

Configuration

Demo

In-memory configuration

JSON configuration

Configuration via command line

Configuration with environment variables

Options pattern

See practical use in AppInsights

https://github.com/rstropek/Samples/tree/master/AspNetCore1Workshop/55-configuration/

Logging

Support for logging built into ASP.NET Core

Various logger built in

- E.g. console, NLog

Details about logging

Consider using Application Insights

- Getting started with AppInsights in ASP.NET Core

Logging

Demo

- JSON file to configure logging
  - .NET Core Logging
  - AppInsights
- Custom logging
- AppInsights portal

https://github.com/rstropek/Samples/tree/master/AspNetCore1Workshop/58-logging/

BASTA!

Dependency Injection

- Support for DI built into ASP.NET Core
  - Details about DI
- Framework-provided services and your own services
- Service Lifetime
  - Transient, Scoped, Singleton, Instance
- Default container can be replaced (details)

Dependency Injection

Demo

Setting up DI

Service Lifetime

https://github.com/rstropek/Samples/tree/master/AspNetCore1Workshop/60-di-scopes/

.NET Core Automation

Test, build, and release automation

CI with .NET Core apps

VSTS supports building and publishing .NET Core apps

- Details

Azure App Services supports .NET Core apps

- Kudu-support for .NET Core

Ready-made Docker image with `Dockerfile`

- `microsoft/dotnet`

Build Automation

Demo

Build and deploy .NET Core in VSTS

https://www.visualstudio.com/en-us/docs/build/apps/aspnet/aspnetcore-to-azure

Dockerfile for .NET Core app

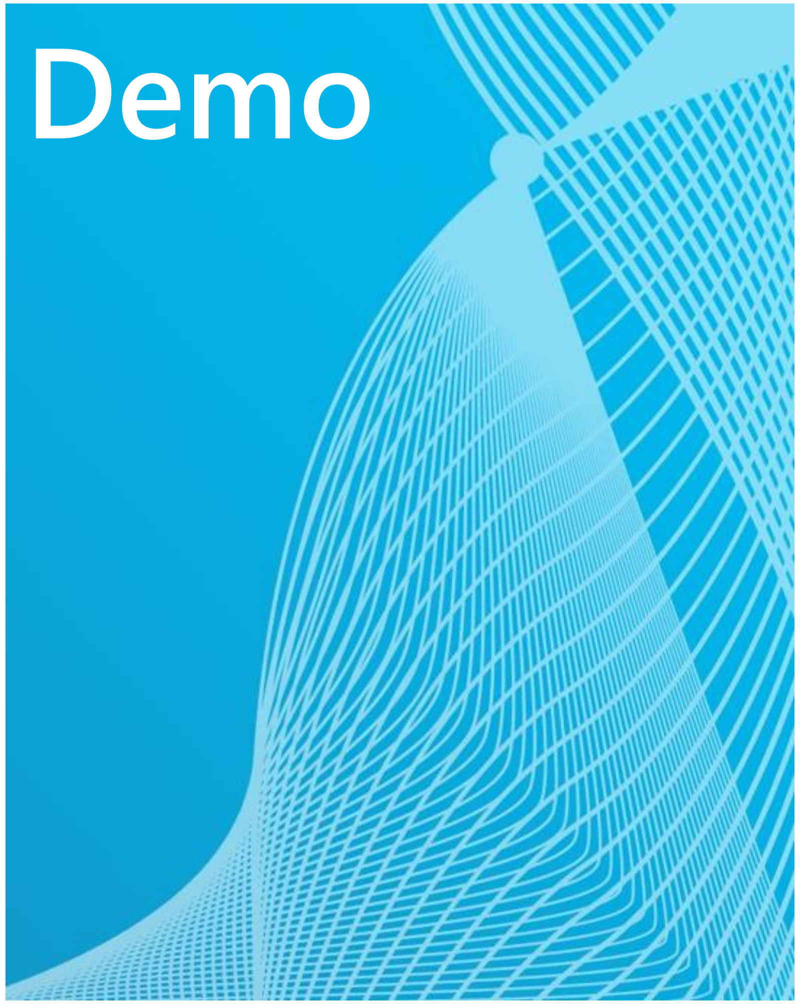
Demo

Unit Testing

.NET Core supports multiple test frameworks

- E.g. XUnit, MSTest
- Compare XUnit and MSTest

Unit Testing



Create and run library with tests

- XUnit (sample)
- MSTest (sample)

Run tests with `VSTest.Console.exe`

```
vstest.console.exe project.json
/UseVsixExtensions:true /logger:trx
```

Project setup

- Folders, `project.json`

https://blogs.msdn.microsoft.com/visualstudioalm/2016/09/01/announcing-mstest-v2-framework-support-for-net-core-1-0-rtm/

C# 6

Tip: Get C# 6 Diagnostic Analyzers, Code Fixes and Refactorings

Auto Properties

```
// default values
public class Customer
{
    public string First { get; set; } = "Jane";
    public string Last { get; set; } = "Doe";
}

// getter only
public class Customer
{
    public string First { get; } = "Jane";
    public string Last { get; } = "Doe";
}

// read only backing fields
public class Customer
{
    public string Name { get; }
    public Customer(string first, string last)
    {
        Name = first + " " + last;
    }
}
```

Expression Bodies

```
// method
public void Print() => Console.WriteLine(First + " " + Last);
```

```
// property
public string Name => First + " " + Last; public Customer
```

```
// indexer
this[long id] => store.LookupCustomer(id);
```

Using Static

```
using static System.Console;
using static System.Math;
using static System.DayOfWeek;
class Program
{
    static void Main()
    {
        WriteLine(Sqrt(3*3 + 4*4));
        WriteLine(Friday - Monday);
    }
}
```

Null Conditional

```
// properties
int? length = customers?.Length; // null if customers is null


// indexers
Customer first = customers?[0]; // null if customers is null


// null conditional - possible Null reference on .Count()
int? first = customers?[0].Orders.Count();
// inline
int? first = (customers != null) ? customers[0].Orders.Count() : null;
// better
int? first = customers?[0].Orders?.Count();


// void
PropertyChanged?.Invoke(this, args);
```

String Interpolation

```
// old
var s = String.Format("{0} is {1} year{{s}} old", p.Name, p.Age);
// new
var s = $"{p.Name} is {p.Age} year{{s}} old";


// format info
var s = $"{p.Name,20} is {p.Age:D3} year{{s}} old";


// expressions
var s = $"{p.Name} is {p.Age} year{(p.Age == 1 ? "" : "s")} old";
```

Nameof

```
if (x == null) throw new ArgumentNullException(nameof(x));



// prints "ZipCode"
WriteLine(nameof(person.Address.ZipCode));
```

Index Initializer

```
var numbers = new Dictionary<int, string> {
    [7] = "seven",
    [9] = "nine",
    [13] = "thirteen"
};
```

Await and Exceptions

```
// Exception filter
try { … }
catch (MyException e) when (myfilter(e))
{
    …
}


// async – await
Resource res = null;
try
{
    res = await Resource.OpenAsync(…);
}
catch(ResourceException e)
{
    await Resource.LogAsync(res, e);
}
finally
{
    if (res != null) await res.CloseAsync();
}
```

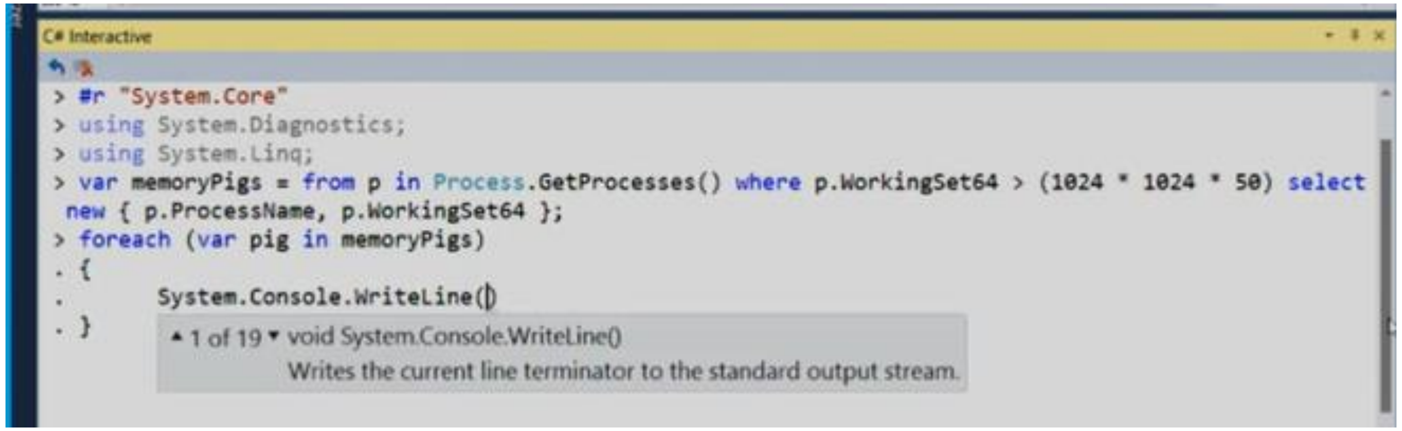
Visual Studio 2015 Updates

C# Scripting

Roslyn Scripting API for C#

- Finally back ;-)
- Sample

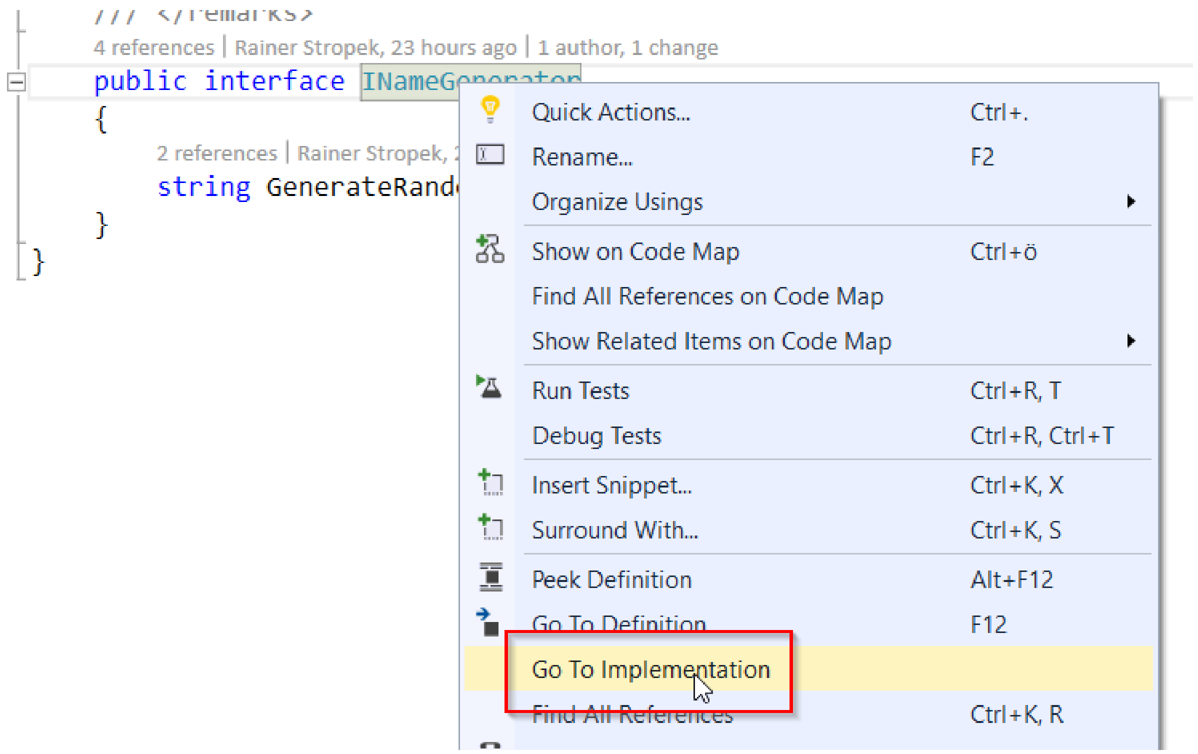
C# Interactive Windows in VS



```
> #r "System.Core"
> using System.Diagnostics;
> using System.Linq;
> var memoryPigs = from p in Process.GetProcesses() where p.WorkingSet64 > (1024 * 1024 * 50) select
new { p.ProcessName, p.WorkingSet64 };
> foreach (var pig in memoryPigs)
. {
.         System.Console.WriteLine()
. }
```

▲ 1 of 19 ▼ void System.Console.WriteLine()
Writes the current line terminator to the standard output stream.

Goto Implementation

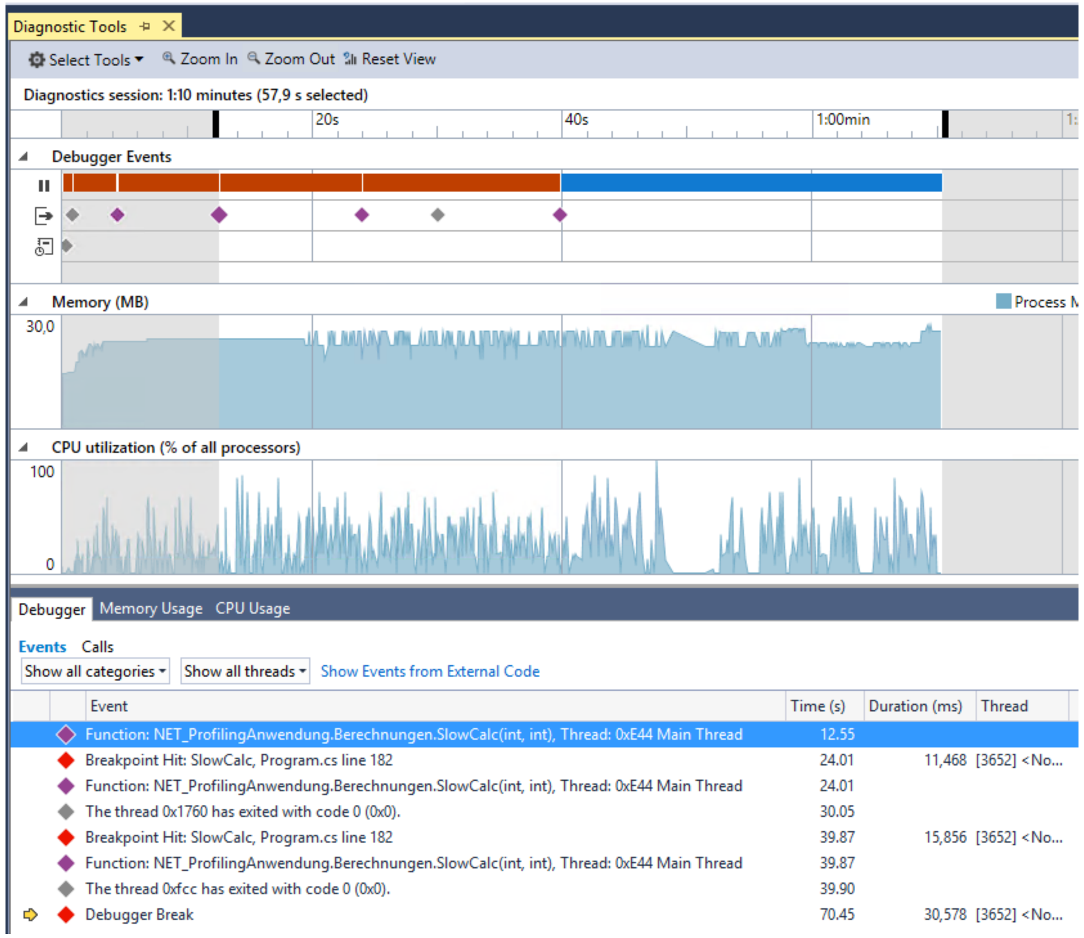


```
/// </remarks>
4 references | Rainer Stropek, 23 hours ago | 1 author, 1 change
public interface INameGenerator
{
        2 references | Rainer Stropek, 2
        string GenerateRand
}
}
```

Quick Actions... Ctrl+.
Rename... F2
Organize Usings
Show on Code Map Ctrl+ö
Find All References on Code Map
Show Related Items on Code Map
Run Tests Ctrl+R, T
Debug Tests Ctrl+R, Ctrl+T
Insert Snippet... Ctrl+K, X
Surround With... Ctrl+K, S
Peek Definition Alt+F12
Go To Definition F12
Go To Implementation
Find All References Ctrl+K, R

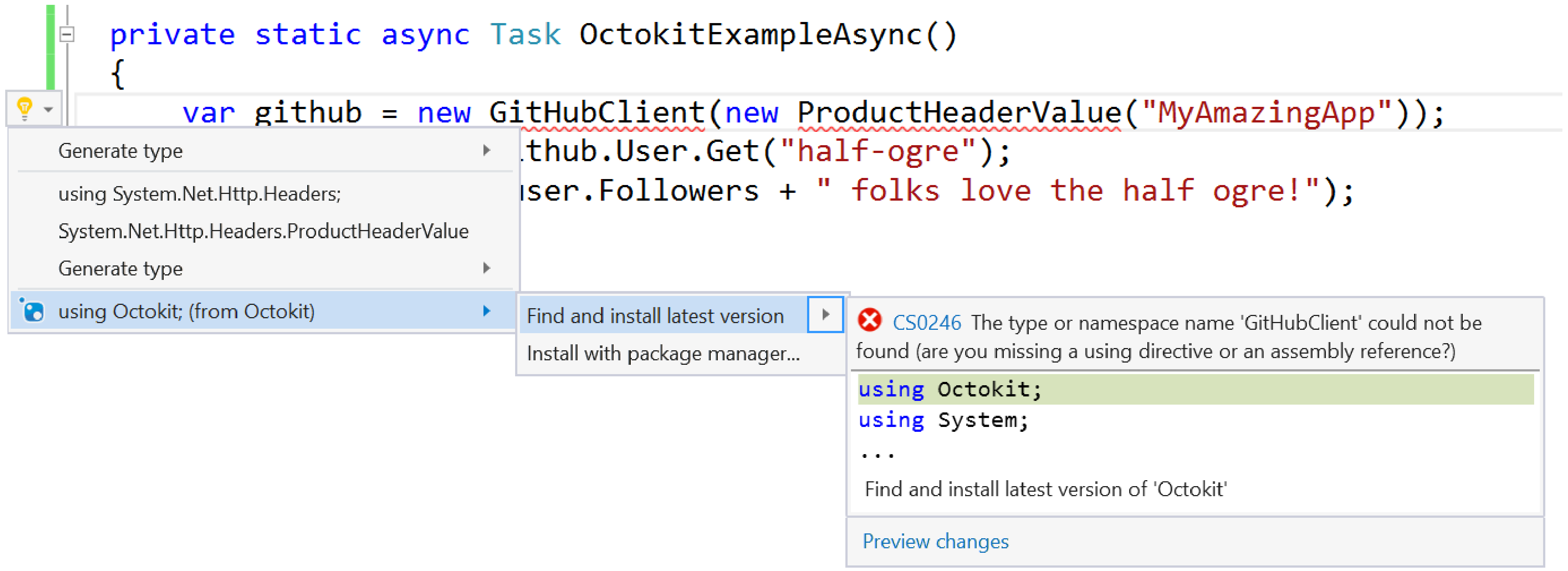
Diagnostic Tools

- Diagnose data on timeline
- IntelliTrace-Events
- Memory usage
  - Incl. GC
- CPU usage



Event	Time (s)	Duration (ms)	Thread
Function: NET_ProfilingAnwendung.Berechnungen.SlowCalc(int, int), Thread: 0xE44 Main Thread	12.55		
Breakpoint Hit: SlowCalc, Program.cs line 182	24.01	11,468	[3652] <No...
Function: NET_ProfilingAnwendung.Berechnungen.SlowCalc(int, int), Thread: 0xE44 Main Thread	24.01		
The thread 0x1760 has exited with code 0 (0x0).	30.05		
Breakpoint Hit: SlowCalc, Program.cs line 182	39.87	15,856	[3652] <No...
Function: NET_ProfilingAnwendung.Berechnungen.SlowCalc(int, int), Thread: 0xE44 Main Thread	39.87		
The thread 0xfcc has exited with code 0 (0x0).	39.90		
Debugger Break	70.45	30,578	[3652] <No...

Add Reference to NuGet Package



```
private static async Task OctokitExampleAsync()
{
    var github = new GitHubClient(new ProductHeaderValue("MyAmazingApp"));
    ...thub.User.Get("half-ogre");
    ...ser.Followers + " folks love the half ogre!");
```

Generate type
using System.Net.Http.Headers;
System.Net.Http.Headers.ProductHeaderValue
Generate type
using Octokit; (from Octokit)
Find and install latest version
Install with package manager...

CS0246 The type or namespace name 'GitHubClient' could not be found (are you missing a using directive or an assembly reference?)

```
using Octokit;
using System;
...
```

Find and install latest version of 'Octokit'

Thank you for coming!

Questions?