Online on-demand Project Support

.NET Core 3.0

Grundlagen

Thorsten Kansy (tkansy@dotnetconsulting.eu)

Thorsten Kansy

Freier Consultant, Softwarearchitekt,
Entwickler, Trainer und Fachautor

DOTNETCONSULTING
by Thorsten Kansy

Agenda

- Verwendete Software
- Einführung
- Konfiguration
- Inversion of Control & Dependency Injection
- Logging
- Deployment
- Migration


pienene
the latvian green studio
pienene
latvian green studio
P
Tikai
ar atļaujām
Nr. 33 - 36
Latviešu
dizaina veikals
& Kafetērija
pienene
CAFÉ SHOP
KAFE VEIKALS
Latvian
design store
& Cafe
Fabisiak
44

Roadmap

.NET Core 3.0 Roadmap

Upcoming Ship Dates

Milestone	Release Date
.NET Core 2.2.x, 2.1.x, 1.x (servicing)	Approximately every 1-2 months or as needed (see also releases)
.NET Core 3.0	Preview releases Final version in second half of 2019 - see Schedule in Preview 3 (from 2019/3). Ship date will be announced at the Build 2019 conference. Previous announcements: original 2018/5 and update 2018/10.

Note: Dates are calendar year (as opposed to fiscal year).

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

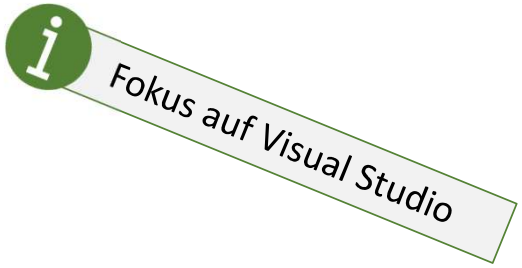
Verwendete Software

Verwendete Software

- Visual Studio 2017 15.9.3+
- .NET Core 2.2

- Visual Studio 2019 16.0+
- .NET Core 3.0

Alternative Editoren



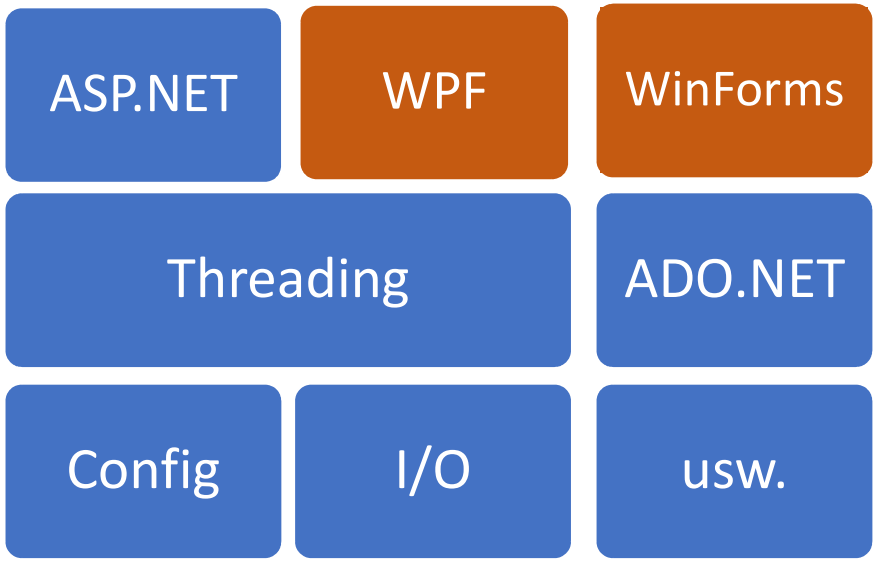
- Windows
  - Visual Studio Code
- MacOS
  - Visual Studio
  - Visual Studio Code
- Linux
  - Visual Studio Code

Und viele mehr, wie z. B. Sublime Text, vi, gedit und Notepad

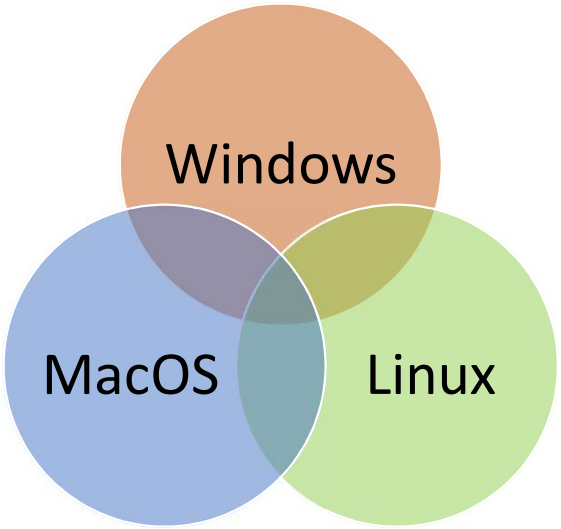
Einführung

.NET-Core-Aufbau

Modular



Multi Plattform



DOTNETCONSULTING
by Thorsten Kansy

.NET Core vs. .NET Framework

.NET Core	.NET Framework
Modularer Aufbau (NuGet)	Systemweites, (fast) monolithisches Framework
WPF, WinForms, ASP.NET, etc.	WPF, WinForms, ASP.NET, etc.
Fast vollständig, nicht überfrachtet	Vollständig, aber überfrachtet
Native Multiplattform (viele Funktionen)	Windows gebundene Plattform
Nur teilweise an Windows gebunden	An Windows gebunden

Was spricht für .NET Core?

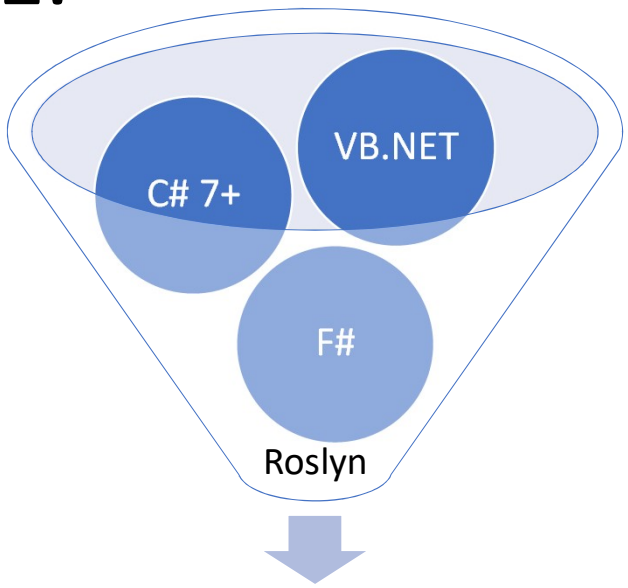
- Multiplattform
  - Windows
  - Linux
  - MacOS
- Höhere Performance
  - Modularität
  - ASP.NET, z. B. Entkopplung von `System.Web.dll`
- Geringerer Ressourcenverbrauch
  - Modularität

Aber was ist mit .NET Standard?

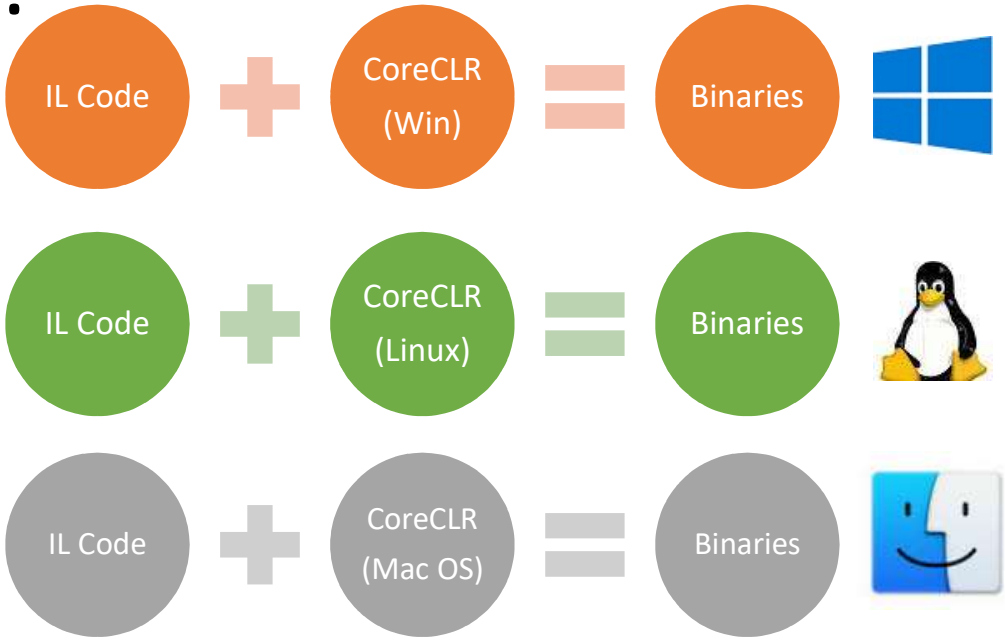
- Gemeinsame Funktionen für alle .NET Branches/Flavors
  - .NET Framework
  - .NET Core
  - Xamarin
- Jeweils in der aktuellen Version
- Erleichtert die Erstellung von Cross-Code

Compile-Prozess(e)

1.



2.



.NET Native

Nativer Code (nur) für Univeral Window Platform (UWP)

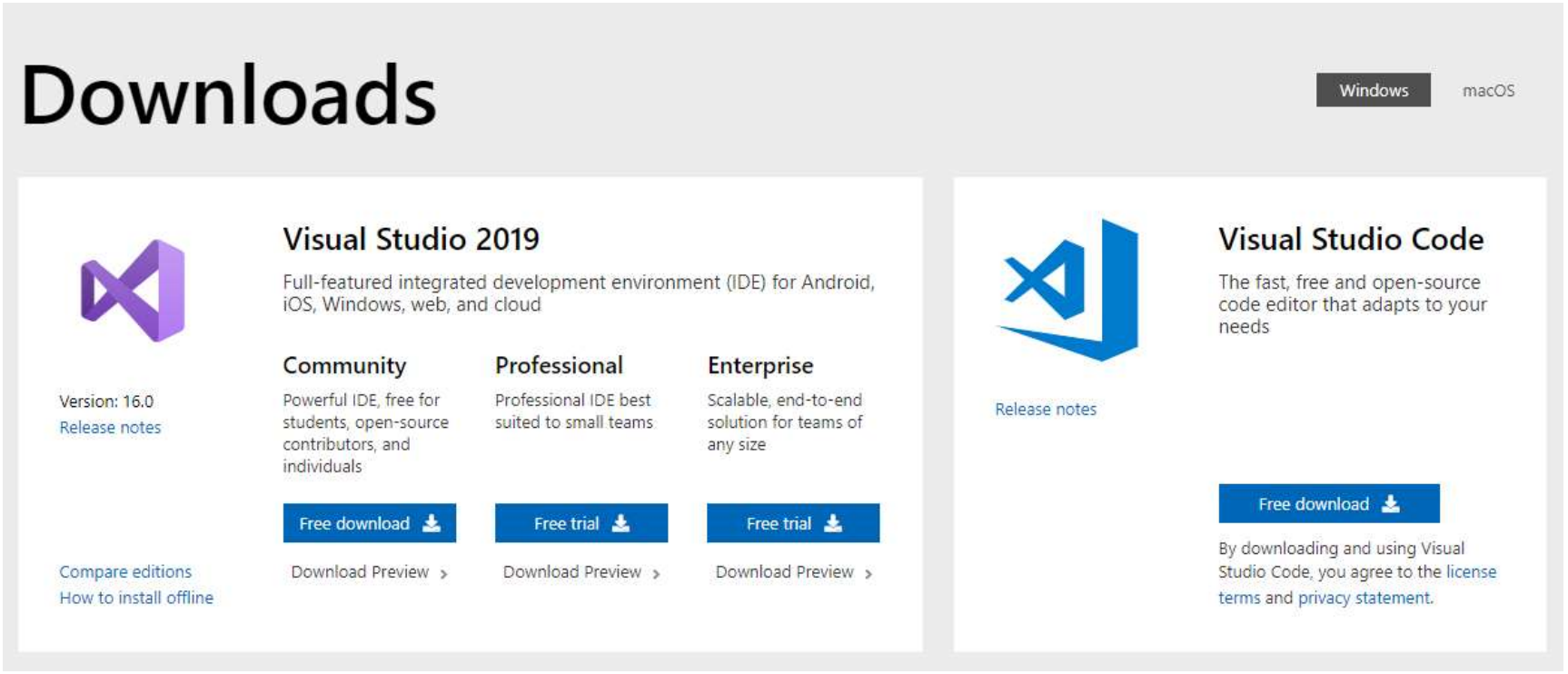
- `ngen.exe` (Native Image Generator)

Vorteile für UWP

- Schnellere Ausführung
- Schnellerer Start
- Optimized app memory usage.

`https://docs.microsoft.com/en-us/dotnet/framework/net-native/`

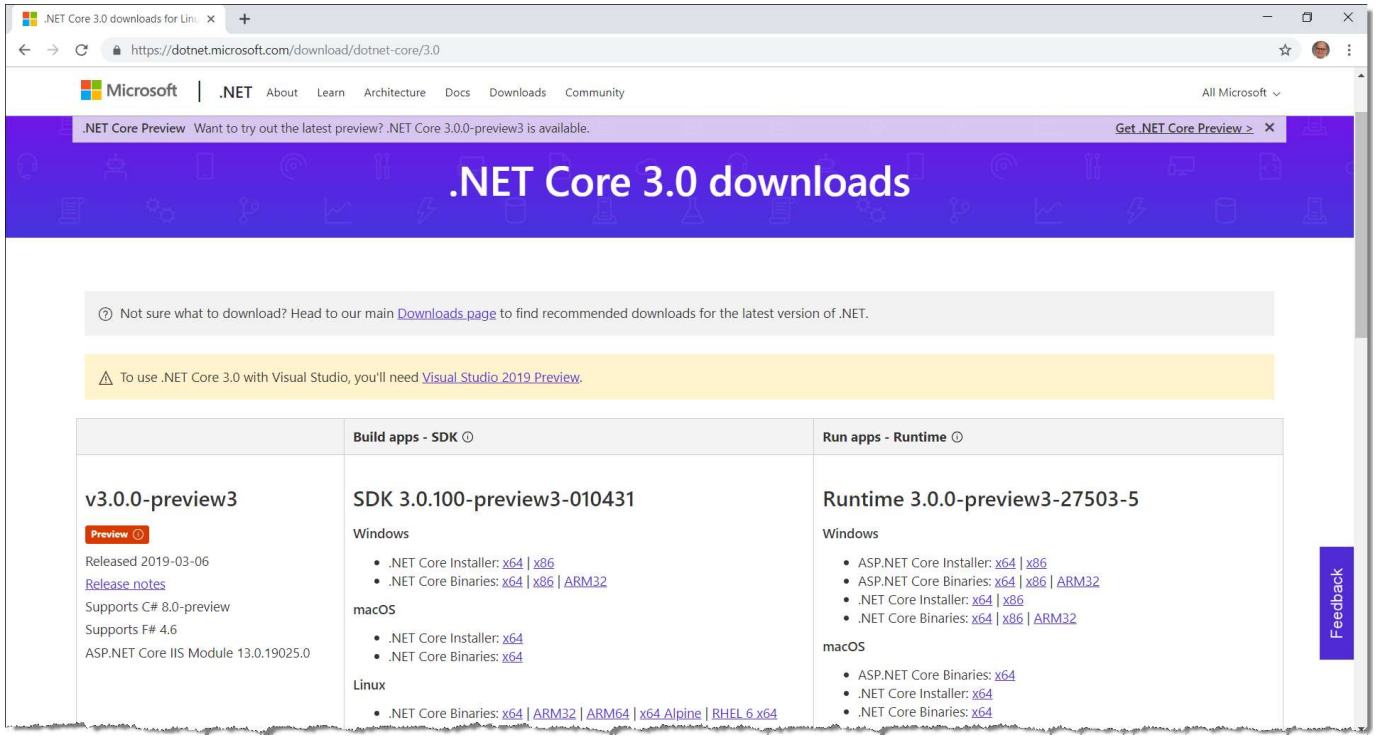
Visual Studio in Stellung bringen



https://www.visualstudio.com/de/downloads

.NET Core in Stellung bringen

- Windows
- Linux
- MacOS



```
https://dotnet.microsoft.com/download/dotnet-core/3.0
```

DOTNETCONSULTING
by Thorsten Kansy

.NET Core CLI

- Command Line für .NET Core
  - Projekterstellung
  - Entity Framework
  - User Secrets (ASP.NET)
  - …

```
dotnet --info
```

```
dotnet --version
```

```
dotnet -list-sdks
```

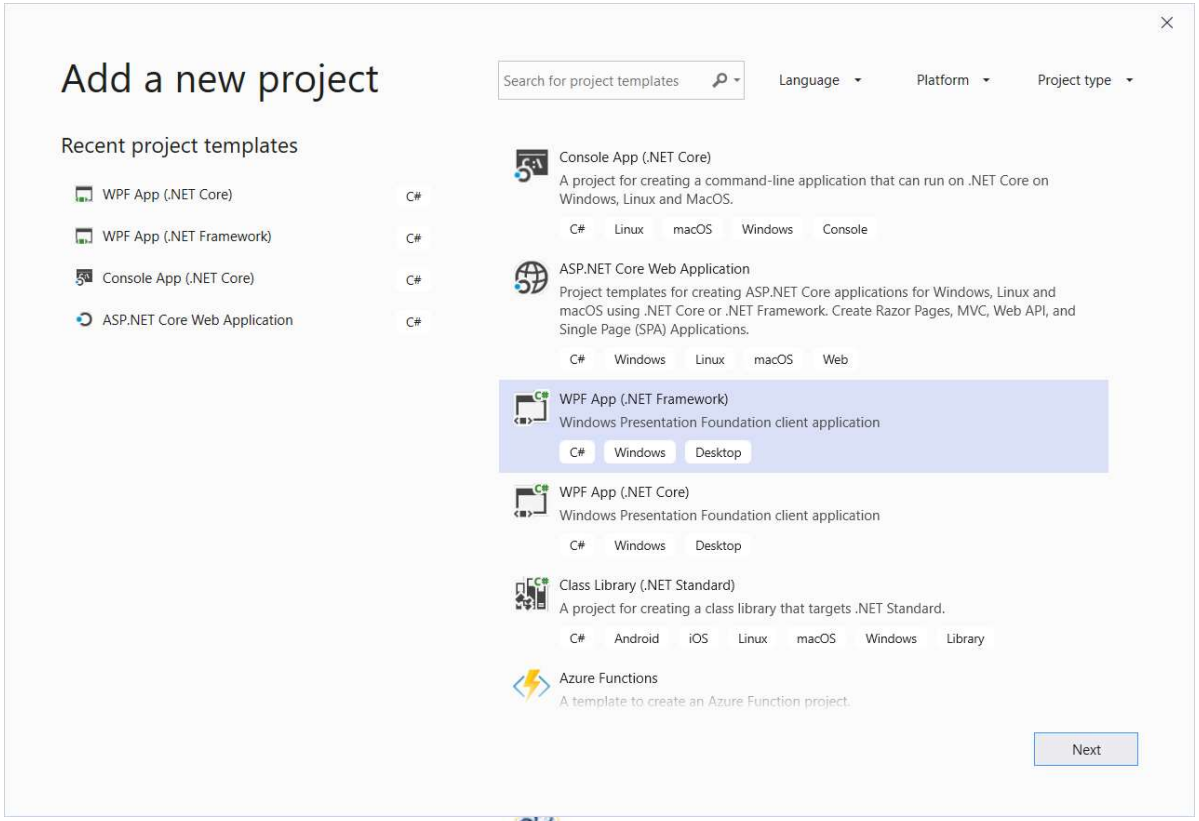
```
dotnet ef
```

„Hallo Welt“ mit CLI und Notepad

```
dotnet new console|WPF|... -n HelloWorld

cd HelloWorld

notepad program.cs

dotnet run [<proj>.csproj]

dotnet <prog>.dll
```


Demo

„Hallo Welt“ mit Visual Studio 2019

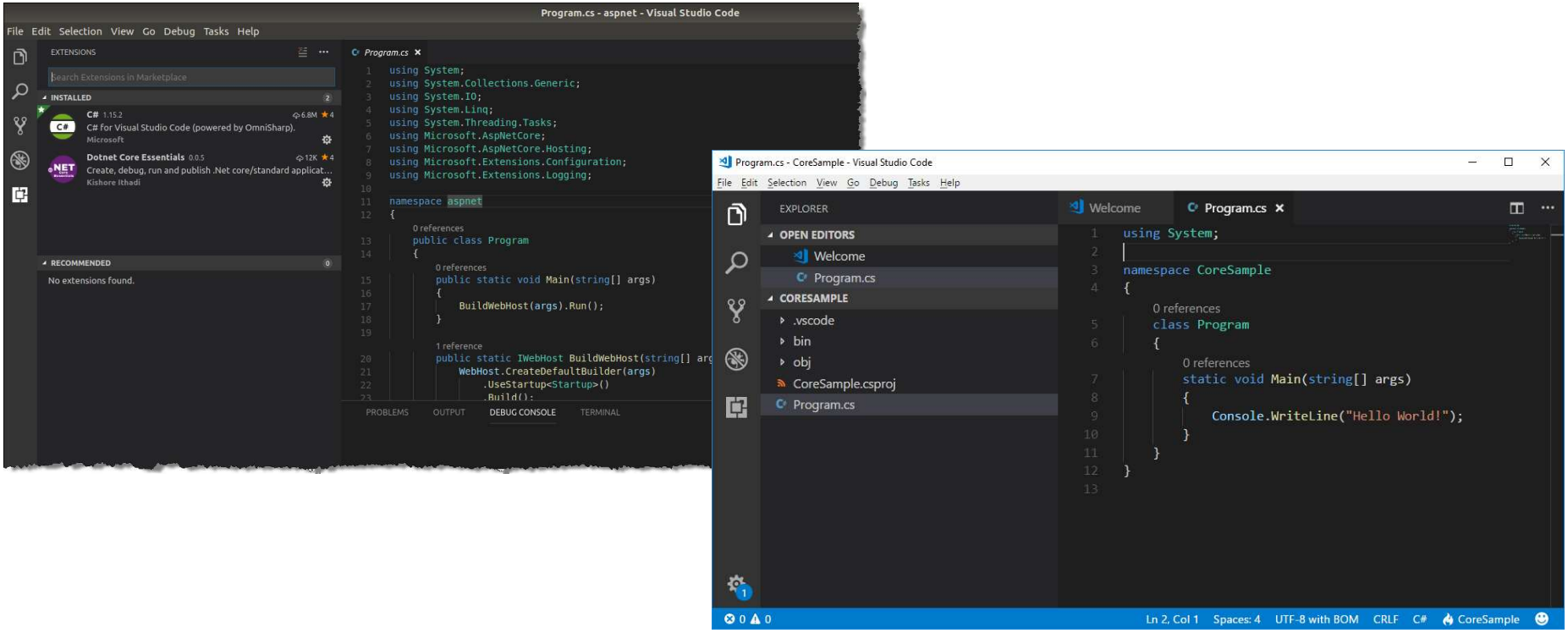



Demo

„Hallo Welt“ mit Visual Studio Code

```
md CoreHW
cd CoreHW
dotnet <prog>.dll
dotnet run <proj>.csproj

code .
```

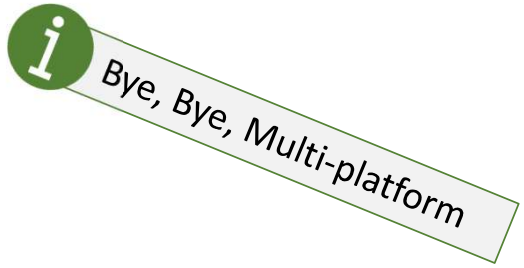
„Hallo Welt“ mit Visual Studio Code (Ubuntu)



DOTNETCONSULTING
by Thorsten Kansy


Demo

Windows Compatibility Pack



- Code Pages
- CodeDom
- Configuration
- Directory Services
- Drawing
- ODBC
- Permissions
- Ports
- Windows Access Control Lists (ACL)
- Windows Communication Foundation (WCF)
- Windows Cryptography
- Windows EventLog
- Windows Management Instrumentation (WMI)
- Windows Performance Counters
- Windows Registry
- Windows Runtime Caching
- Windows Services

```
https://docs.microsoft.com/en-us/dotnet/core/porting/windows-compat-pack
```

Multi-Plattform-Code

Einsatz mit Guarding

```
if (RuntimeInformation.IsOSPlatform(OSPlatform.Windows))
{
    // Windows Code, Zugriff auf Registry
    using (RegistryKey key = Registry.CurrentUser.OpenSubKey(@"Software\Fabrikam\AssetManagement"))
    {
        if (key?.GetValue("LoggingDirectoryPath") is string configuredPath)
            Console.WriteLine(configuredPath);
    }
}
else if (RuntimeInformation.IsOSPlatform(OSPlatform.Linux))
{
    // Linux Code
    // ...
```

```
Install-Package Microsoft.Windows.Compatibility
```

```
Install-Package Microsoft.DotNet.Analyzers.Compatibility
```


Demo

Konfiguration

Konfigurationsprovider

Provider	NuGet-Paket
JSON-Datei	`Microsoft.Extensions.Configuration.Json`
XML-Datei	`Microsoft.Extensions.Configuration.Xml`
INI-Datei	`Microsoft.Extensions.Configuration.Ini`
Umgebungsvariablen	`Microsoft.Extensions.Configuration.EnvironmentVariables`
Programmargumente	`Microsoft.Extensions.Configuration.CommandLine`
In-Memory-Collection	–
Azure Key Vault	`Microsoft.Extensions.Configuration.AzureKeyVault`
User Secrets (nur ASP.NET Core)	`Microsoft.Extensions.Configuration.UserSecrets`
Custom	–

```
Install-Package Microsoft.Extensions.Configuration.Json (s.o.)
```

Konfiguration verwenden

```
// Konfiguration vorbereiten, Provider nach Bedarf anfügen
IConfigurationBuilder builder = new ConfigurationBuilder()
    // Umgebungsvariablen hinzufügen
    .AddEnvironmentVariables()
    // Json-Datei hinzufügen
    .AddJsonFile("Config.json");

// Konfiguration abschließen, Werte einlesen
IConfigurationRoot config = builder.Build();

// Zugriff
string windowHeight = config["App:Window:Height"];
```


Demo

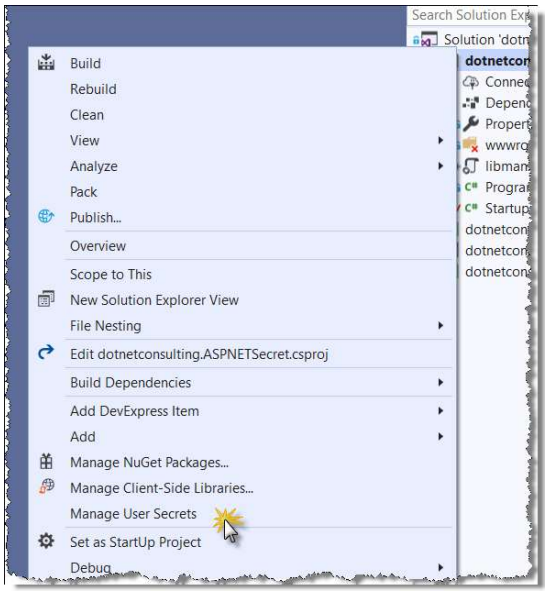
ASP.NET User Secrets

Sensible Einstellungen auslagern

```
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
<UserSecretsId>576504d1-3c17-4fc0-b0a0-23ee841be7e6</UserSecretsId>
  </PropertyGroup>
</Project>
```

Betriebssystem	Speicherort
Windows	`%APPDATA%\microsoft\UserSecrets\<userSecretsId>\secrets.json`
Linux	`~/.microsoft/usersecrets/<userSecretsId>/secrets.json`
MacOS	`~/.microsoft/usersecrets/<userSecretsId>/secrets.json`

User Secrets verwalten



Aktion	.NET Core CLI
User Secret setzen	`dotnet user-secrets set Geheimnis "My Name is Bond"`
User Secrets auflisten	`dotnet user-secrets list`
User Secret löschen	`dotnet user-secrets remove Geheimnis`
Alle User Secrets löschen	`dotnet user-secrets clear`

DOTNETCONSULTING
by Thorsten Kansy


Demo

Eigenen Custom Provider schreiben

Code-Klasse	Aufgabe
`SqlConfigurationExtensions`	Erweiterung der Fluent API
`SqlDatabaseConfigurationSource`	Implementiert `IConfigurationSource` , um eine Konfiguration mittels Fluent API zu ermöglichen.
`SqlDatabaseConfigurationProvider`	Implementiert `IConfigurationProvider` und stellt die eigentliche Programmlogik dar, die die Konfigurationswerte bereitstellt.
`SqlDatabaseChangeToken`	Implementiert `IChangeToken` und bietet die Möglichkeit, Veränderungen bei den Konfigurationswerten zu signalisieren. Hier nicht weiter vertieft, da die Änderungen zwischen dem Lesen zwei zusammenhängender Werte auftreten können.


Demo

Entity Framework Core konfigurieren

Nicht in `DbContext.OnConfiguring(...)`

```
private static IServiceProvider createDependencyInjectionContainer(IConfigurationRoot Configuration)
{
    ...
     // EF Context konfigurieren
     .AddDbContext<SamplesContext1>(
         o => o
         .UseLazyLoadingProxies()
         .UseSqlServer(Configuration["ConnectionStrings:EFConString"])
         .ConfigureWarnings(w => w.Throw(RelationalEventId.QueryClientEvaluationWarning))
         .EnableSensitiveDataLogging(true)
     )
    ...
}
```


Demo

Eigene Komponenten richtig konfigurieren

Beispiel „SmtpService“

- `SmtpServiceConfig`

  Konfiguration mit konkreten Werten, Quelle beliebig
- `SmtpServiceConfigBuilder`

  Validiert und erzeugt die Konfiguration
- `SmtpServiceCollectionExtensions`

  Stellt `AddSmtpService()`-Methode bereit

Eigene Komponenten richtig konfigurieren

```
static private void ConfigureServices(IServiceCollection serviceCollection)
{
    …
    // SMTP-Service konfigurieren, ggf. je nach Environment
    serviceCollection.AddSmtpServer<MockedSmtpService>(o =>
    {
        // Konkrete Werte aus Konfigration, Quelle jedoch beliebig und hier unwichtig
        o.SetHostAndPort("192.168.1.1", 25);
        o.Sender = "produktion@dotnetconsultng.eu";
    });
    …
}
```


Demo

Inversion of Control & Dependency Injection

Inversion of Control und Dependency Injection

- Überschaubarer Code (Single Responsibility Principle)
- Wiederverwendbarkeit
- Besser testbarer Code

`IServiceCollection` IoC-Container

`IServiceProvider` DI-Service

```
Install-Package Microsoft.Extensions.DependencyInjection
```

```
Install-Package Microsoft.Extensions.DependencyInjection.Abstractions
```

IoC-Konfiguration

```
static private void ConfigureServices(IServiceCollection serviceCollection)
{
    // Instanz pro GetService<>()-Aufruf/ GetRequiredService<>()-Aufruf
    serviceCollection.AddTransient<IOrderService, SnailMailOrderService>();

    // ODER Instanz als Singelton
    serviceCollection.AddSingleton<IOrderService, SnailMailOrderService>();

    // ODER Instanz per (ASP.NET)-Request (siehe CreateScope())
    serviceCollection.AddScoped<IOrderService, SnailMailOrderService>();
}
```

Lebenszeit von Instanzen

Variante	Lebenszeit
`AddTransient()`	Der IoC-Container liefert jedes Mal eine neu erzeugte Instanz
`AddScoped()`	Während einer Anfrage (z. B. an einen ASP.NET-Controller) wird die gleiche Instanz geliefert. Bei der nächsten Anfrage wird wiederum eine neue Instanz geliefert
`AddSingleton()`	Der IoC-Container liefert jedes Mal die gleiche Instanz zurück

DI-Service

```
IServiceProvider services = serviceCollection.BuildServiceProvider();

// Liefert NULL, wenn der Service nicht geliefert werden kann
services.GetService<IOrderService>()
    ?.PlaceOrder("Wattestäbchen", 10);

// Wirft eine Exception (System.InvalidOperationException)
// statt NULL zu liefern
services.GetRequiredService<IOrderService>()
    .PlaceOrder("Wattestäbchen", 10);
```

DI-Scope

```
// Scope erzeugen (via ASP.NET Core als Request)
using (IServiceScope scope = services.CreateScope())
{
    ISmtpService smtpServiceScope =
        scope.ServiceProvider.GetService<ISmtpService>();
    IOrderService orderServiceScope =
        services.GetService<IOrderService>();

    // ...
}
```

Gilt für: MVC und Web-API

DI-Service – Controller

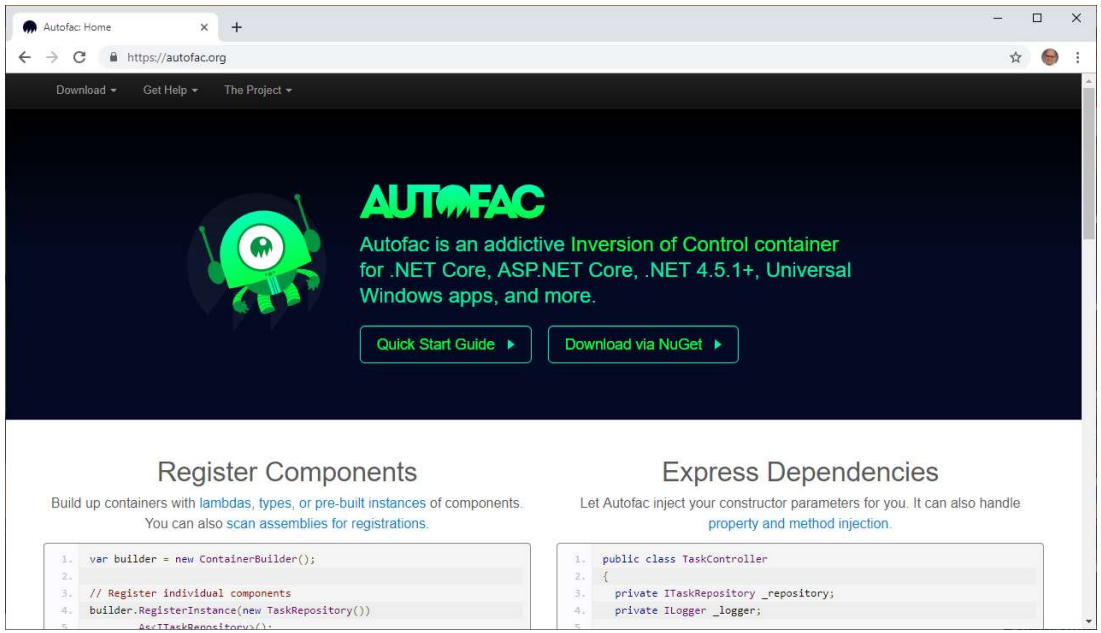
```
public class HomeController : Controller
{
    private readonly IOrderService _orderService;

    public HomeController(IOrderService OrderService)
    {
        _orderService = OrderService;
    }

    public IActionResult Index()
    {
        // _orderService verwenden
    }
}
```

DI-Service – Razor Pages

```
public class IndexModel : PageModel
{
    private readonly IOrderService _orderService;

    public IndexModel(IOrderService OrderService)
    {
        _orderService = OrderService;
    }

    public void OnGet()
    {
        // _orderService verwenden
    }
}
```


Demo

IoC- und DI-Alternative – Autofac



http://autofac.org

```
Install-Package Autofac
```

```
Install-Package Autofac.Extentions.DependencyInjection
```

Autofac-Setup

```
public IServiceProvider ConfigureServices(IServiceCollection services)
{
    // Andere Framework-Services hinzufügen
    // services.AddMvc();

    // Autofac hinzufügen und konfigurieren
    ContainerBuilder containerBuilder = new ContainerBuilder();
    containerBuilder.RegisterModule<DefaultModule>();
    containerBuilder.Populate(services);
    IContainer container = containerBuilder.Build();

    return new AutofacServiceProvider(container);
}
```

```
Install-Package Autofac
```

```
Install-Package Autofac.Extentions.DependencyInjection
```

Autofac-Konfiguration

```
public class DefaultModule : Module
{
    protected override void Load(ContainerBuilder builder)
    {
        builder.RegisterType<EMailOrderService>()
            .As<IOrderService>()
            // Konstruktor-Parameter
            .WithParameter("SmtpHost", "127.0.0.1")
            // Eigenschaft
            .WithProperty("Sender", "tkansy@dotnetconsulting.eu");
    }
}
```


Demo

Logging

Logging

- 3 gute Gründe:
- Fehlersuche, Fehlersuche & Fehlersuche!

`ILoggerFactory` Logger Factory

`ILogger` Logger Service

```
Install-Package Microsoft.Extensions.Logging
```

```
Install-Package Microsoft.Extensions.Logging.Abstractions
```

Logging-Provider

Level	NuGet-Paket
Konsole	`Microsoft.Extensions.Logging.Console`
Debugger-Monitor (Output window)	`Microsoft.Extensions.Logging.Debug`
Trace Listener (Output window)	`Microsoft.Extensions.Logging.TraceSource`
Windows-Ereignisprotokoll	`Microsoft.Extensions.Logging.EventLog`
EventSource/EventListener	`Microsoft.Extensions.Logging.EventSource`
Azure-App-Dienste „Diagnoseprotokolle“ und „Log Stream“	`Microsoft.Extensions.Logging.AzureAppServices`
Datei, Datenbank, SMTP etc.	`Serilog.* // NLog.* // usw.`
Custom	–

Logging konfigurieren

```
static void LoggingDemo1(string scopeName = null)
{
    // Logger Factory konfigurieren
    ILoggerFactory loggerFactory = new LoggerFactory()
        .AddConsole(LogLevel.Trace, true)
        .AddDebug();

    ILogger logger = loggerFactory.CreateLogger<Program>();

    demoLogging(logger, scopeName);
}
```


Demo

Logging-Level

Variante	Lebenszeit
`Trace (0)`	Ablaufverfolgung, die auch sensible Informationen enthalten kann
`Debug (1)`	Technische Debug-Informationen für die Fehlersuche
`Information (2)`	Nachverfolgung des allgemeinen Ablaufs der Anwendung
`Warning (3)`	Warnungen bei ungewöhnlichen oder unerwarteten Ereignissen
`Error (4)`	Fehler und Ausnahmen, die vom Code nicht behandelt werden können
`Critical (5)`	Für Fehler, die sofortige Aufmerksamkeit erfordern

```
...public enum LogLevel
{
    ...Trace = 0,
    ...Debug = 1,
    ...Information = 2,
    ...Warning = 3,
    ...Error = 4,
    ...Critical = 5,
    ...None = 6
}
```


Demo

Logging-Scopes

Bereich im Logging zwecks besserer Leßbarkeit

```
using (logger.BeginScope($"Scope y = {y}"))
{
    for (int z = 0; z < 2; z++)
    {
        logger.LogDebug($"y = {y}, z = {z}");
    }
}
```


Demo

Logging in Action I

```
logger.Log(LogLevel.Trace, "Trace-Log");
logger.Log(LogLevel.Debug, "Debug-Log");
logger.Log(LogLevel.Information, "Information-Log");

logger.LogWarning("Warning-Log");
logger.LogError("Error-Log");
logger.LogCritical("Critical-Log");
```

Logging in Action II

```
// Wenn etwas schief geht
try
{
    throw new Exception("Passierschein A38 nicht gefunden.");
}
catch (Exception ex)
{
    logger.LogError(ex, "Error");
    throw;
}
```

Logging in Action III

```
// Wenn mal etwas richtig schief geht
try
{
    throw new Exception("Passierschein A38 nicht vorhanden.");
}
catch (Exception ex)
{
    logger.LogCritical(ProgramEventId.PermitNotFound, ex,
                                                "Permit A38 not found ");
    throw;
}
```

Logging in Action IV

```
// Oder wenn etwas nicht dort ist, wo es sein sollte
string importFilename = @"c:\nix\data.csv";
try
{
    string import = File.ReadAllText(importFilename);
}
catch (Exception ex)
{
    logger.LogCritical(ProgramEventId.ImportFileFailed, ex,
                       "Import file '{0}' not ", importFilename);
}
```


Demo

EventId

```
public static class ProgramEventId
{
     public static readonly EventId CriticalSituation201
            = new EventId(201, "CriticalSituation201");

     public static readonly EventId PermitNotFound
            = new EventId(202, "PermitNotFound");

     public static readonly EventId ImportFileFailed
            = new EventId(203, "ImportFileFailed");
}
```


Demo

Logging & Dependency Injection

```
// Factory
serviceCollection.AddSingleton<ILoggerFactory, LoggerFactory>();

// Generischer Logger
serviceCollection.AddSingleton(typeof(ILogger<>), typeof(Logger<>));

public class AppController
{
    private readonly ILogger<AppController> _logger;

    public Application(ILogger<AppController> Logger)
    {
        _logger = Logger;
    }
}
```


Demo

Logging in Datei – Serilog

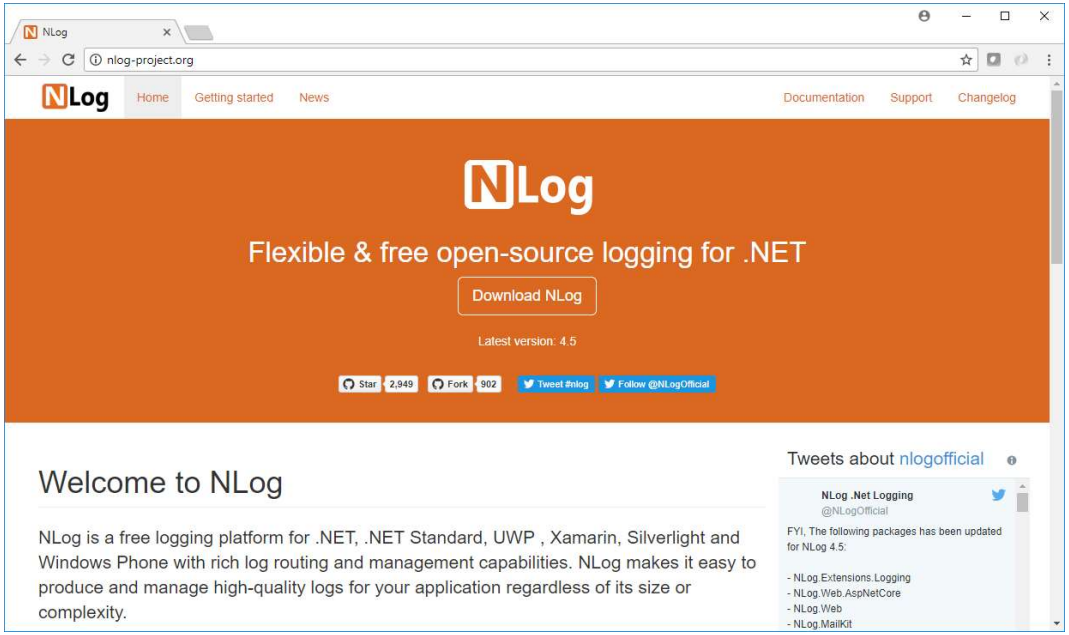
```
// Filename fürs Logging
string fileName = Path.Combine(AppContext.BaseDirectory, "Logging.txt");

// Logger Factory konfigurieren;
ILoggerFactory loggerFactory = new LoggerFactory()
    .AddFile(fileName, LogLevel.Information);

ILogger logger = loggerFactory.CreateLogger<Program>();
```

https://github.com/serilog/

```
Install-Package Serilog.Extensions.Logging.File
```


Demo

Nlog – Fully-featured Logging Framework



```
http://nlog-project.org
```

```
Install-Package NLog
```

```
Install-Package NLog.Extensions.Logging
```


Demo

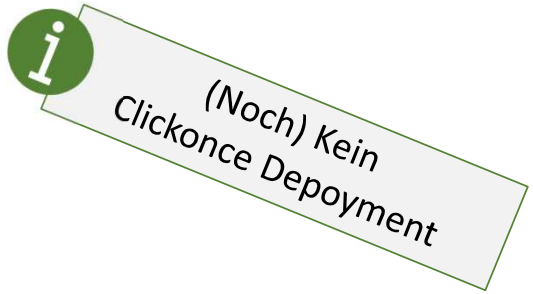
Eigenen Custom Provider schreiben

Code-Klasse	Aufgabe
`ColorConsoleLoggingExtentions`	Erweiterung der Fluent API
`ColorConsoleLogger`	Der eigentliche Logger, der das `Ilogger`-Interface implementiert.
`ColorConsoleLoggerConfiguration`	Die Konfiguration die der Logger verwendet. Dies ist ein schlichtes POCO
`ColorConsoleLoggingProvider`	Der Provider, der die benötigten Instanzen des Loggers erzeugt und das `ILoggerProvider`-Interface implementiert.


Demo

Allgemeines Deployment

Deployment



- Visual Studio 2019: Publish
  - Framework-dependent Deployment (FDD)
  - Self-contained Deployment (SCD)
- App Bundler für SCD
  - (Noch nicht verfügbar)
- Visual Studio 2019: Pack
  - Nuget-Paket erstellen
- .NET Core CLI

Visual Studio 2019: Publish

- FDD
  - Project -> Publish
- SCD
  - Project -> Publish

2.0

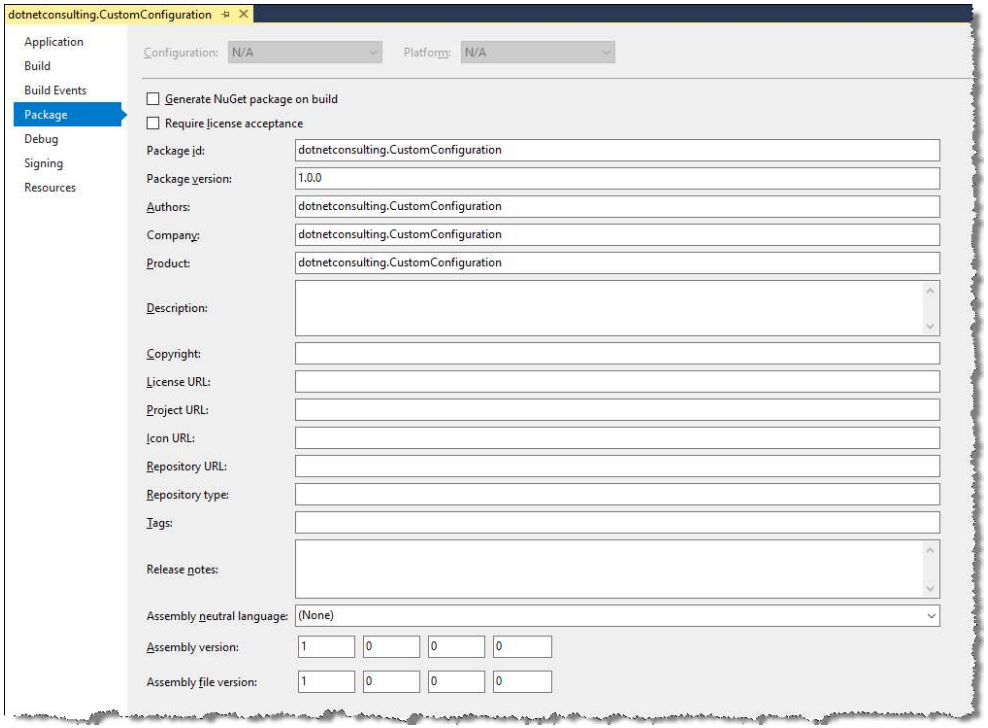
```
== *.csproj-Daten ==
<PropertyGroup>
  <RuntimeIdentifiers>win10-x64;osx.10.11-x64;linux-x64</RuntimeIdentifiers>
</PropertyGroup>
```

https://docs.microsoft.com/en-us/dotnet/core/deploying/deploy-with-vs


Demo

Visual Studio 2019: Pack

- Erstellt ein NuGet-Paket
  - `nuget.exe` aus VS heraus
- Details:
  - Projekt -> Properties -> Package



DOTNETCONSULTING
by Thorsten Kansy


Demo

.NET Core CLI

Framework-dependent Deployment

```
dotnet publish -c Release
```

Self-contained Deployment

```
dotnet publish -c Release -r win10-x64
dotnet publish -c Release -r ubuntu.16.10-x64
dotnet publish -c Release -r osx.10.11-x64
```


Demo

Migration

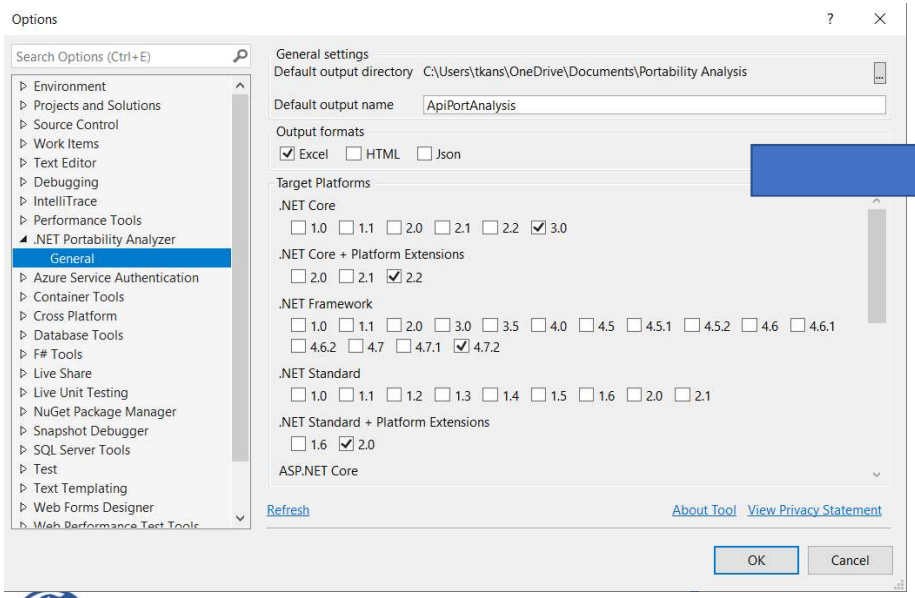
Migration zu .NET Core

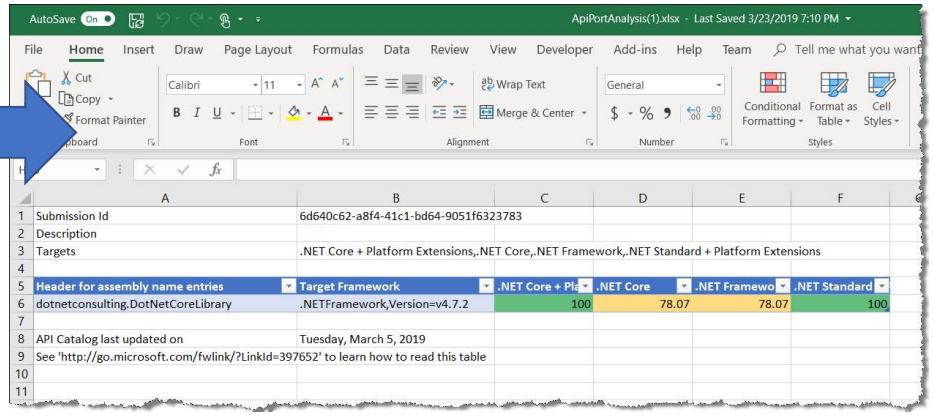
1. Aufwand abschätzen mit Analyzers
   - API Portability Analyzer Tool
   - NET Core 3.0 Desktop Api Analyzer
2. Dritthersteller (UI-)Komponenten prüfen
3. Projekte auf .NET Framework 4.6.2+ migrieren
4. Testportierung
5. …

```
https://github.com/Microsoft/dotnet-apiport/
```

.NET Portability Analyzer

Visual Studio Plug-in für 2015/ 2017/ 2019



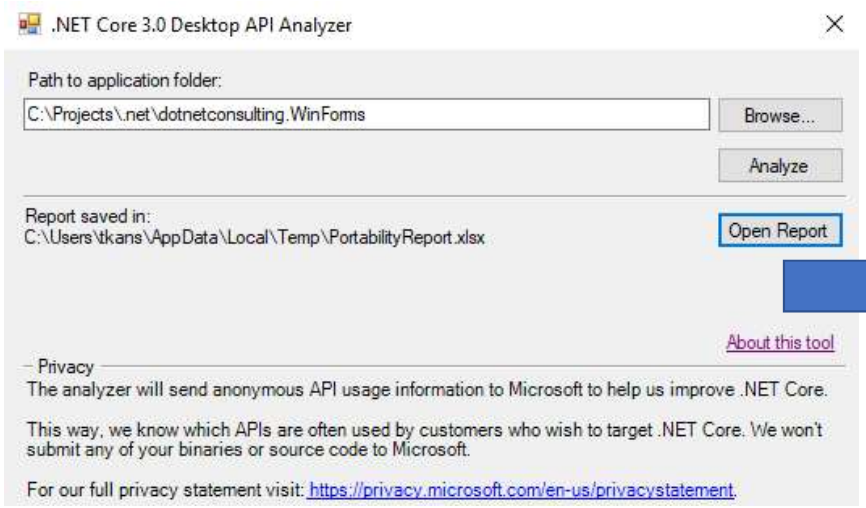


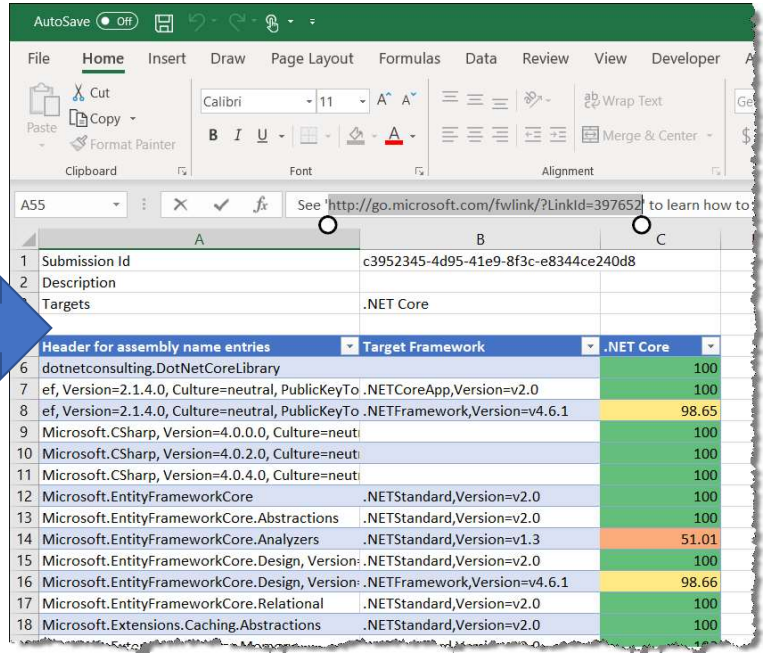
https://docs.microsoft.com/en-us/dotnet/standard/analyzers/portability-analyzer

DOTNETCONSULTING
by Thorsten Kansy


Demo

.NET Core 3.0 Desktop Api Analyzer





	A	B	C
1	Submission Id	c3952345-4d95-41e9-8f3c-e8344ce240d8	
2	Description		
3	Targets	.NET Core	
	Header for assembly name entries	Target Framework	.NET Core
6	dotnetconsulting.DotNetCoreLibrary		100
7	ef, Version=2.1.4.0, Culture=neutral, PublicKeyTo	.NETCoreApp,Version=v2.0	100
8	ef, Version=2.1.4.0, Culture=neutral, PublicKeyTo	.NETFramework,Version=v4.6.1	98.65
9	Microsoft.CSharp, Version=4.0.0.0, Culture=neuti		100
10	Microsoft.CSharp, Version=4.0.2.0, Culture=neuti		100
11	Microsoft.CSharp, Version=4.0.4.0, Culture=neuti		100
12	Microsoft.EntityFrameworkCore	.NETStandard,Version=v2.0	100
13	Microsoft.EntityFrameworkCore.Abstractions	.NETStandard,Version=v2.0	100
14	Microsoft.EntityFrameworkCore.Analyzers	.NETStandard,Version=v1.3	51.01
15	Microsoft.EntityFrameworkCore.Design, Version=	.NETStandard,Version=v2.0	100
16	Microsoft.EntityFrameworkCore.Design, Version=	.NETFramework,Version=v4.6.1	98.66
17	Microsoft.EntityFrameworkCore.Relational	.NETStandard,Version=v2.0	100
18	Microsoft.Extensions.Caching.Abstractions	.NETStandard,Version=v2.0	100

https://devblogs.microsoft.com/dotnet/are-your-windows-forms-and-wpf-applications-ready-for-net-core-3-0


Demo

Fragen?

Links

https://www.visualstudio.com/de/downloads

https://github.com/ElectronNET/Electron.NET

https://github.com/AvaloniaUI/Avalonia

https://github.com/mellinoe/ImGui.NET