



Noser Engineering .NET 9 und C# 13

23.01.2025

«Zusammen
MehrWert»



1

Hans Peter Bornhauser

Software Architect, Consultant and
Trainer

CTO at Noser Engineering AG
Digicomp Trainer since 1997
Microsoft Certified Trainer since 2003



hanspeter.bornhauser@noser.com



2

.NET 9

- Released in Nov. 2024
- STS license (Standard time support = 1.5 years)
- Some “Highlights”
 - Performance
 - BinaryFormatter no longer supported

3 24.01.2025 WE KNOW HOW.



3

Upgrading Existing Projects

1. [Check Breaking Changes](#)
2. Update Visual Studio
3. Update Project files (find & replace)

TẮS GÊT G SẮN XỘ S L Ậ T Ậ T Ậ T G SẮN XỘ S L
4. (Restart Visual Studio)
5. Update NuGet Packages
6. Test



4 24.01.2025 WE KNOW HOW.



4

Experience?

Upgrade	5 min for solution with > 30 projects – no issues
Performance	overall up to 40% faster
Cool Features	??
My favorite	field keyword (currently in preview) Guid V7

5 24.01.2025 WE KNOW HOW.



5



6

params Collections

How we all know it: `public void ShowNumbersArray(params int[] numbers)`

Call: `ShowNumbersArray(1, 2, 3, 4);`

C# 13: `public void ShowNumbersEnumerable(params IEnumerable<int> numbers)`

Works now with

- `Span<T>`
- types that implement `IEnumerable<T>`
- Interfaces that define `Add()` methods

7 24.01.2025 WE KNOW HOW.



7

Partial Properties

What we know since C# 2: **Partial methods**

Normal usage: Declaration in tool generated code

`partial void OnFontSizeChanging(int oldValue, int newValue);`

No return type, implicit private

C# 13:

`[ObservableProperty] // CommunityToolkit.Mvvm to support XAML binding`
`public partial int FontSize { get; private set; }`

8 24.01.2025 WE KNOW HOW.



8

5

Lock Object

Before: `private readonly object _lock = new();`

.NET 9: `private readonly Lock _lock = new();`

```
using (_lock.EnterScope())
{
    // synchronized code
}
```

```
lock (_lock)
{
    // synchronized code
}
```

11 24.01.2025 WE KNOW HOW.



11

Index access in initializers

```
public int[] Numbers { get; set; } = new int[5];
```

```
Numbers =
{
    [^1] = 10, // Last element
    [^2] = 20, // Second-to-last element
    [^3] = 30, // Third-to-last element
    [^4] = 40, // Fourth-to-last element
    [^5] = 50 // Fifth-to-last element
}
```

12 24.01.2025 WE KNOW HOW.



12

More C# 12 Features

- [ÔwêstlôăđRêşỘlựtjîỘnRsîỘsîtjỠ]
- New escape sequence \e for escape character ESC (e.g. for clear screen)
- ref struct interfaces

.NET 9

Base Framework

15

NOSER
ENGINEERING

Task.WhenEach

WhenAll

```
var allResults = await Task.WhenAll(computationTasks);

foreach (var result in allResults)
{
    Console.WriteLine($"Task {result.Id} finished in {result.Delay} ms.");
}
```

WhenEach

```
var computationTaskList = computationTasks.ToList();
await foreach (var task in Task.WhenEach(computationTaskList))
{
    var result = await task;
    Console.WriteLine($"Task {result.Id} finished in {result.Delay} ms.");
}
```

17 24.01.2025 WE KNOW HOW.



17

Guid Version 7

Time ordered Guid's

Version 4 Guid:

```
var guid4 = Guid.NewGuid();
```

Version 7 Guid:

```
var guid7 = Guid.CreateVersion7();
```

GUID: 01945b8f-f4bf-7014-b873-2b57a7294ee9

GUID: 01945b8f-f4bf-7532-97f5-15b012f584d0

GUID: 01945b8f-f4bf-7e8f-8442-13f7277a4888

GUID: 01945b8f-f4bf-7c50-a645-6edeb7354501

18 24.01.2025 WE KNOW HOW.



18

JSON

Customizing indentation

Customizing indentation

```

    JsonSerializerOptions options = new JsonSerializerOptions
    {
        Indent = "  ",
        WriteIndented = true
    };

```

Default options for Web API

```

JsonSerializerOptions webOptions = JsonSerializerOptions.Web;
JsonSerializer.Serialize(new { Value = 42 }, JsonSerializerOptions.Web);

```

19 24.01.2025 WE KNOW HOW.



19

Null checks

Non-Nullable reference types which are null, thrown an exception.

```

JsonSerializerOptions options = new() { RespectNullableAnnotations = true };

```

Global Feature Switch

```

<ItemGroup>
  <RuntimeHostConfigurationOption
    Include="System.Text.Json.Serialization.RespectNullableAnnotationsDefault"
    Value="true" />
</ItemGroup>

```

Performance

- ~30% faster serialization than .NET 8

20 24.01.2025 WE KNOW HOW.



20

More JSON News

Export JSON Schema

```
JsonSerializerOptions options = JsonSerializerOptions.Default;
JsonNode schema = options.GetJsonSchemaAsNode(typeof(Person));
```

Custom Enum serialisation: JsonStringEnumMemberName

```
enum ErrorCode
{
    None = 0,
    Unknown = 1,
    [JsonStringEnumMemberName("Connection Lost")]
    ConnectionLost = 100,
    [JsonStringEnumMemberName("Outlier Reading")]
    OutlierReading = 200
}
```

21 24.01.2025 WE KNOW HOW.



21

More .NET News

- Generic Ordered Dictionary: `OrderedDictionary<TKey, TValue>`

22 24.01.2025 WE KNOW HOW.



22

ASP.NET Core



23

ASP.NET Core 9 New Features

1. Static File Management
2. OpenAPI support
3. HybridCache API
4. Improved Developer Exception Page and Debugging
5. Support for Keyed Dependency Injection in Middleware
6. SignalR improvements
7. Performance Minimal API vs. NET 8:
 - 15% more requests / second
 - 93% less memory

24

Static File Management

- Replace `app.UseStaticFiles()` => `app.MapStaticAssets()`
- Etag to detect unchanged files
- Compression with brotli, ~10x smaller
- Files need to be known at development time, works only after publish
- For MVC
 - Use `WithStaticAssets()`
- For Blazor
 - Use `@Assets["lib/bootstrap/dist/css/bootstrap.min.css"]` for includes
 - Add `<ImportMap />` to Html-Head section

25 24.01.2025 WE KNOW HOW.



25

New OpenAPI

- Replace Swashbuckle


```
<PackageReference Include="Microsoft.AspNetCore.OpenApi" Version="9.0.1" />
<PackageReference Include="Swashbuckle.AspNetCore" Version="7.2.0" />
```

```
builder.Services.AddOpenApi();
app.MapOpenApi();
```
- For UI: we still use SwashBuckle


```
app.UseSwaggerUI(options => options.SwaggerEndpoint("/openapi/v1.json", "v1"));
```
- No support for XML comments

26 24.01.2025 WE KNOW HOW.



26

HybridCache (preview)

Combination of In-memory Cache and Distributed Cache
Registration

```
builder.Services.AddHybridCache();
builder.Services.AddHybridCache(options =>
{
    options.DefaultEntryOptions = new HybridCacheEntryOptions
    {
        Expiration = TimeSpan.FromSeconds(30),
        LocalCacheExpiration = TimeSpan.FromSeconds(30)
    };
});
```

27 24.01.2025 WE KNOW HOW.



27

Usage of HybridCache

```
public class ProductService(HybridCache _cache) : IProductService
{
    public async Task<Product> GetProductAsync(string productId) =>
        // Use HybridCache to fetch data or create it if missing
        await _cache.GetOrCreateAsync(
            $"prod:{productId}", // unique cache key
            async entry => await GetProduct(productId));
}
```

28 24.01.2025 WE KNOW HOW.



28

Support for Keyed DI in Middleware

```
builder.Services.AddKeyedSingleton<MySingletonClass>("test");
builder.Services.AddKeyedScoped<MyScopedClass>("test2");

var app = builder.Build();
app.UseMiddleware<MyMiddleware>();
app.Run();
```

29 24.01.2025 WE KNOW HOW.



29

Middleware with Keyed Services

```
internal class MyMiddleware
{
    public MyMiddleware(RequestDelegate next,
        [FromKeyedServices("test")] MySingletonClass singletonService)
    {
        _next = next;
        _singletonService = singletonService;
    }

    public Task Invoke(HttpContext context,
        [FromKeyedServices("test2")] MyScopedClass scopedService)
    {
        // Middleware logic here
        return _next(context);
    }
}
```

30 24.01.2025 WE KNOW HOW.



30

Blazor News

- .NET MAUI Blazor and Web App solution template
- Detect render mode, interactivity and location (important with prerendering)
- Constructor injection in components
- InputNumber supports type="range"
- Missing: support for PWA on project creation

31 24.01.2025 WE KNOW HOW.



31

Constructor injection in components

- Before (property injection)

```

    private void ConfigureServices(IServiceCollection services)
    {
        services.Add<MyComponent>();
    }

```

- Now constructor injection

```

    private void ConfigureServices(IServiceCollection services)
    {
        services.Add<MyComponent>();
    }

```

32 24.01.2025 WE KNOW HOW.



32

EF Core

33

EFCore Performance



NOSER
ENGINEERING

34

Performance EF Core 9 vs. Dapper

EF Core

```
public async Task<IList<FactResellerSale>> EFCoreToListWithLinq(int orderDateKey)
{
    return await _dbContext.FactResellerSales
        .Where(frs => frs.OrderDateKey == orderDateKey)
        .ToListAsync();
}
```

Dapper

- 1.04 mal schneller (5.6ms)
- 1.54x mehr Memory (1.46 MByte)

```
public async Task<IList<FactResellerSale>> DapperToList(int orderDateKey)
{
    var qry = await _dbConnection.QueryAsync<FactResellerSale>(
        sql: "SELECT * FROM FactResellerSales WHERE OrderDateKey = @OrderDateKey",
        new { OrderDateKey = orderDateKey });
    return qry.ToList();
}
```

Quelle: <https://trailheadtechnology.com/ef-core-9-vs-dapper-performance-face-off/>

35 24.01.2025 WE KNOW HOW.



35

What's new?

- Azure Cosmos DB improvements
- AOT and pre-compiled queries (experimental)
- LINQ and SQL translation => performance
- Auto-Compiled models
 - Before: command-line tool: `dotnet ef dbcontext optimize`
 - Now: entry in project file

RsôřêstjÿĜsôuř

ÉGÔřġîñîcêCộղġêyġ ħjsuê ÉGÔřġîñîcêCộղġêyġ

ÉGŞçăġġôĭđNộđêĭşġăġê Ćuîĭđ ÉGŞçăġġôĭđNộđêĭşġăġê

RsôřêstjÿĜsôuř

36 24.01.2025 WE KNOW HOW.



36



37 24.01.2025 WE KNOW HOW.

