DEVNOT DOTNET KONFERANSI 2024

.NET 8 New Features with Code Examples

Mert Metin

ABOUT ME

Mert Metin

- Senior Software Engineer SOMPO
- Blogger

mrtmtn

mertmtn

Blogcu Mühendis

_mertmetin

QR to reach me

Agenda

01

C# 12

- Primary Constructors
- Collection Expressions
- Spread Operator
- Data Annotations
- Default Values in Lambda Exp.
- Exprerimental
- Random.GetItems
- Shuffle
- Alias Any Type

02

Keyed DI Services

Enhanced Feature for .NET Dependency Injection

03

Performance Improvements

- FrozenSet
- CompositeFormat
- SearchValues<T>
- Benchmark Results* of the Improvements

*All results have created by BenchmarkDotnet tool

Primary Constructors

- Simple class creation
- Looks like method parameter

```
public class Product
{
    0 references
    public Product(string name)
    {

    }

    0 references
    public string Name { get; set; }
    0 references
    public string Description { get; set; }
    0 references
    public string Category { get; set; }
    0 references
    public decimal Price { get; set; }
}
```

Typical constructor before C# 12

```
public class Product(string name)
{
    0 references
    public Product(string name, string price) : this("")
    {

    }

    0 references
    public string Name { get; set; }
    0 references
    public string Description { get; set; }
    0 references
    public string Category { get; set; }
    0 references
    public decimal Price { get; set; }
}
```

Primary Constructor

- Useful for dependency injection based projects

- Reduces code lines
- Increases readability

```
public class RegisterManager : IRegisterService
{
    private readonly IStudentService _studentService;
    private readonly IRegisterRepository _registerRepository;
    private readonly IUserService _userService;

    0 references
    public RegisterManager(
        IStudentService studentService,
        IRegisterRepository registerRepository,
        IUserService userService)
    {
        _studentService = studentService;
        _registerRepository = registerRepository;
        _userService = userService;
    }
}
```

```
public class RegisterManager(
    IStudentService studentService,
    IRegisterRepository registerRepository,
    IUserService userService) : IRegisterService
{
    private readonly IStudentService _studentService = studentService;
    private readonly IRegisterRepository _registerRepository = registerRepository;
    private readonly IUserService _userService = userService;
}
```

Collection Expressions

Initialization and defining operations are more concise and clear.

```
//Before C# 12

List<int> numberListOldWay = new List<int>() { 0, 2, 4, 6, 8 };
List<int> numberEmptyListOldWay = new();

int[] numberArrayOldWay = new int[] { 1, 3, 5, 7, 9 };
int[] emptyArrayOldWay = Array.Empty<int>();

//With C# 12

List<int> numberListNewWay = [0, 2, 4, 6, 8];
List<int> numberEmptyListNewWay = [];

int[] numberArrayNewWay = [1, 3, 5, 7, 9];
int[] emptyArrayNewWay = [];

List<int> numberOldWay = new List<int>() { 1, 2, 3, 4, 5 };
```

Collection initialization can be simplified

Suppress or configure issues

IDE0028 Collection initialization can be simplified

Lines 43 to 45

```
-List<int> numberOldWay = new List<int>() { 1, 2, 3, 4, 5 }
+List<int> numberOldWay = [1, 2, 3, 4, 5];
```

Spread Operator

Merge, clone, extend collections with two dots (..)

Reminds us of Javascript

```
string[] vowels = ["a", "e", "i", "o", "u"];

string[] consonants = ["b", "c", "d", "f", "g", "h", "j", "k", "l", "m",
                       "n", "p", "q", "r", "s", "t", "v", "w", "x", "z"];

string[] turkishAlphabet = ["ç", "ü", "ö", "ğ", .. vowels, .. consonants,];

Console.Write(string.Join(",", turkishAlphabet));
```

Seç Microsoft Visual Studio Debug Console

```
ç,ü,ö,ğ,a,e,i,o,u,b,c,d,f,g,h,j,k,l,m,n,p,q,r,s,t,v,w,x,z
```

Data Annotations

Added new attributes

- Length
- Range
- AllowedValues
- DeniedValues
- Base64String

```
0 references
public class Person
{
    0 references
    public int Id { get; set; }

    //(MinLength,MaxLength)
    [Length(5, 15)]
    0 references
    public string Name { get; set; }

    0 references
    public string Surname { get; set; }

    ///<summary>
    ///[0,150) --MinimumIsExclusive =false, MaximumIsExclusive = true  -> 0-149
    ///<br></br>[0,150] --MinimumIsExclusive =false, MaximumIsExclusive = false -> 0-150
    ///<br></br>(0,150) --MinimumIsExclusive =true, MaximumIsExclusive = true   -> 1-149
    ///</summary>
    [Range(0, 150, MinimumIsExclusive = false, MaximumIsExclusive = false)]
    0 references
    public int Age { get; set; }

    [DeniedValues("X", "Y")]
    0 references
    public string Gender { get; set; }

    [AllowedValues("TR")]
    0 references
    public string CountryCode { get; set; }

    [Base64String]
    0 references
    public string Picture { get; set; }
}
```

Default Values in Lambda Exp.

Before C# 12

```
var greeting = (string firstName, string lastName = "") =>
            Console.WriteLine($"Thanks, {firstName}
```

CS9058: Feature 'lambda optional parameters' is not available in C# 11.0. Please use language version 12.0 or greater.

Current

default parameter

```
var greeting = (string firstName, string lastName = "") =>
    Console.WriteLine($"Thanks, {firstName} {lastName} for downloading .NET 8");

greeting("Mert");
greeting("Mert", "Metin");
```

Output

```
Thanks, Mert  for downloading .NET 8
Thanks, Mert Metin for downloading .NET 8
```

Experimental Attribute

An attribute which specifies as experimental feature on method or class

```
[Experimental("UpdatePerson")]
1 reference
void UpdatePerson(Person person)
{
    _context.Database.ExecuteSqlRaw("UPDATE Persons SET Name='Mert'");
}

UpdatePerson(person);
```

void UpdatePerson(Person person)

UpdatePerson: 'UpdatePerson(Dotnet8NewFeatures.DataAnnotations.Person)' is for evaluation purposes only and is subject to change or removal in future updates. Suppress this diagnostic to proceed.

Belongs to System.Diagnostics.CodeAnalysis

Experimental Attribute

Suppress the warnings

```
<PropertyGroup>
  <TargetFramework>net8.0</TargetFramework>
  <NoWarn>UpdatePerson</NoWarn>
</PropertyGroup>
```

Source Code

```
#pragma warning disable UpdatePerson
UpdatePerson(person);
#pragma warning restore UpdatePerson
```

Using Pragma Blocks

Random.GetItems<T>

Selects a specific number of items from a given set of elements randomly

Drawback is;

Resulting random collection can include duplicate entries.

```
string[] vowels = ["a", "e", "i", "o", "u"];
string[] consonants = ["b", "c", "d", "f", "g", "h", "j", "k", "l", "m",
                       "n", "p", "q", "r", "s", "t", "v", "w", "x", "z"];
string[] alphabet = [.. vowels, .. consonants, "y"];

var selectedItems = Random.Shared.GetItems(alphabet, 5);

foreach (var item in selectedItems)
{
    Console.WriteLine(item);
}
```

Output

```
u
m
g
l
v
```

Unexpected Output

```
i
i
b
d
s
```

Random.GetItems<T>

Flagging mechanisim using dictionary to avoid duplications

```
T[] GetItems<T>(T[] choices, int length)
{
    ArgumentNullException.ThrowIfNull(choices);
    ArgumentOutOfRangeException.ThrowIfNegative(length);

    var destination = new T[length];
    var choicesDictionary = choices.Distinct().ToDictionary(x => x, x => false);

    var i = 0;
    while (destination.Length != choicesDictionary.Count(x => x.Value))
    {
        var element = choices[Random.Shared.Next(choices.Length)];

        if (!choicesDictionary[element])
        {
            destination[i] = element;
            choicesDictionary[element] = true;
            i++;
        }
    }
    return destination;
}
```

Shuffle

Performs an in-place shuffle of an array.

Useful in some cases;

machine learning,
gambling,
online exam

Belongs to Random class

Shuffle - Code Examples

```
string[] players =
    [
        "Mert",    "Ali",    "Ahmet",    "İbrahim",    "İlkan",
        "Hasan",    "Erman",    "Onur",    "Muzaffer",    "Aydın",
        "Ümit",     "Sebastian", "Yiğit",   "Ayhan"
    ];
```

```
Before Shuffle
----------
Mert
Ali
Ahmet
İbrahim
İlkan
Hasan
Erman
Onur
Muzaffer
Aydın
Ümit
Sebastian
Yiğit
Ayhan
```

```
Random.Shared.Shuffle(players);
```

```
players.ToList().ForEach(x => Console.WriteLine(x));
```

```
After Shuffle
----------
Onur
Ahmet
Aydın
Mert
İbrahim
Ümit
Hasan
Sebastian
Erman
Ali
Ayhan
İlkan
Yiğit
Muzaffer
```

Alias Any Type

Creating custom type with “using” keyword.

“global using” are required for using everywhere

```
global using DotnetVersionHistory = (
    string Version,
    string ReleaseType,
    System.DateOnly LatestReleaseDate,
    System.DateOnly? EndOfSupport
    );
```

```
List<DotnetVersionHistory> GetDotnetVersionHistoryList()
{
    return [
        new DotnetVersionHistory(".NET 9.0","Standard Term Support",new DateOnly(2024,4,11),null),
        new DotnetVersionHistory(".NET 8.0","Long Term Support",new DateOnly(2024,4,9),new DateOnly(2026,11,10)),
        new DotnetVersionHistory(".NET 7.0","Standard Term Support",new DateOnly(2024,4,9),new DateOnly(2024,5,14)),
        new DotnetVersionHistory(".NET 6.0","Long Term Support",new DateOnly(2024,4,9),new DateOnly(2024,11,12)),
        ];
}
```

Keyed DI Services

- Naming services with key.
- Same service - different named services or different lifetimes

Key, could be string or enum type

```
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddKeyedTransient<IStudentService, JuniorService>(nameof(StudentType.Junior));
builder.Services.AddKeyedTransient<IStudentService, SeniorService>(nameof(StudentType.Senior));
builder.Services.AddKeyedTransient<IStudentService, SpecialStudentService>(nameof(StudentType.Special));
builder.Services.AddKeyedTransient<IStudentService, SophomoreService>(nameof(StudentType.Sophomore));
```

(extension) IServiceCollection IServiceCollection.AddKeyedTransient<IStudentService, SophomoreService>(object? serviceKey) (+ 1 overload)
Adds a transient service of the type specified in IStudentService with an implementation type specified in SophomoreService to the specified IServiceCollection.

Returns:
A reference to this instance after the operation has completed.

Program.cs service declaration

Keyed DI Services - Usages

```
builder.Services.AddKeyedTransient<IStudentService, SpecialStudentService>(nameof(StudentType.Special));
builder.Services.AddKeyedSingleton<IStudentService, SophomoreService>(nameof(StudentType.Sophomore));
```

Program.cs service declaration

```
0 references
public IActionResult IndexFromKeyed([FromKeyedServices(nameof(StudentType.Special))] IStudentService _specialStudentService,
                                    [FromKeyedServices(nameof(StudentType.Sophomore))] IStudentService _sophomoreStudentService)
{

    ViewBag.Service1 = _specialStudentService.GetStudentId.ToString();
    ViewBag.Service2 = _sophomoreStudentService.GetStudentId.ToString();

    return View();
}
```

FromKeyedServices - Given key implemented on ActionMethod

```
public class StudentFactory(IServiceProvider provider) : IStudentFactory
{
    private readonly IServiceProvider _provider = provider;

    4 references
    public IStudentService GetService(string serviceName)
    {
        return serviceName switch
        {
            nameof(StudentType.Junior) => _provider.GetKeyedService<IStudentService>(nameof(StudentType.Junior)),
            nameof(StudentType.Senior) => _provider.GetKeyedService<IStudentService>(nameof(StudentType.Senior)),
            nameof(StudentType.Special) => _provider.GetKeyedService<IStudentService>(nameof(StudentType.Special)),
            nameof(StudentType.Freshman) => _provider.GetKeyedService<IStudentService>(nameof(StudentType.Freshman)),
            nameof(StudentType.Sophomore) => _provider.GetKeyedService<IStudentService>(nameof(StudentType.Sophomore)),
            "FreshmanSingleton" => _provider.GetRequiredKeyedService<IStudentService>("FreshmanSingleton"),
            _ => throw new NotImplementedException($"Could not find any service for {serviceName}"),
        };
    }
}
```

Factory Method

```
public class StudentController(IStudentFactory serviceFactory) : Controller
{
    0 references
    public IActionResult IndexServiceProvider()
    {
        var freshmanStudent = serviceFactory.GetService(nameof(StudentType.Freshman));
        var seniorStudent = serviceFactory.GetService(nameof(StudentType.Senior));

        ViewBag.Service1 = seniorStudent.GetStudentId.ToString();
        ViewBag.Service2 = freshmanStudent?.GetStudentId.ToString();
        return View();
    }
}
```

Implementation

FrozenSet<T>

- Frozen means immutable
- Impossible to manipulation - add,clear,remove operations

```
Frozen Set Elements
1 2 3 4 5 6
Counts
 numbers:6
 immutableNumbers:6
 addedImmutableNumbers:7
 frozenSet:6
```

```
List<int> numbers = [1, 2, 3, 4, 5];
numbers.Add(6);

ImmutableList<int> immutableNumbers = numbers.ToImmutableList();
ImmutableList<int> addedImmutableNumbers = immutableNumbers.Add(7);

FrozenSet<int> frozenSet = numbers.ToFrozenSet();
frozenSet.Add(8);

Console.WriteLine("Frozen Set Elements");
foreach (var number in numbers)
    Console.Write($"{number} ");
```

FrozenSet<T> Benchmark Results

Method	Mean
CreateList	623.1 ns
CreateFrozen	25,220.2 ns
CreateHashSet	10,853.7 ns
CreateImmutableList	15,100.0 ns
LookupList	62,752.8 ns
LookupFrozen	3,259.4 ns
LookupHashSet	6,541.8 ns
LookupImmutableList	2,475,883.4 ns

- Slower creation
  because of modification the set
- Faster lookup or reading
  because of immutable set

```
public class LookupBenchmark
{
    private const int Iterations = 1000;
    private FrozenSet<int> frozenSet = Enumerable.Range(0, Iterations).ToFrozenSet();

    [Benchmark]
    0 references
    public void CreateFrozen()
    {
        var result = Enumerable.Range(0, Iterations).ToFrozenSet();
    }


    [Benchmark]
    0 references
    public void LookupFrozen()
    {
        for (var i = 0; i < Iterations; i++)
            _ = frozenSet.Contains(i);
    }
}
```

CompositeFormat

Allows to parse a composite format string once and reuse this instance multiple times. This approach significantly enhances performance, when the format string has created dynamically but used repeatedly.

```
[MemoryDiagnoser]
1 reference
public class CompositeFormatBenchmark
{
    private readonly string name = "Mert";
    private readonly string surname = "Metin";
    private static readonly string format = "Hello {0} {1}!";

    private readonly CompositeFormat GreetingFormat = CompositeFormat.Parse(format);

    [Benchmark]
    0 references
    public string StringFormat_String() => string.Format(CultureInfo.InvariantCulture, format, name, surname);

    [Benchmark]
    0 references
    public string CompositeFormat_String() => string.Format(CultureInfo.InvariantCulture, GreetingFormat, name, surname);

}
```

Method	Mean	Error	StdDev	Gen0	Allocated
StringFormat_String	108.44 ns	2.247 ns	6.338 ns	0.0088	56 B
CompositeFormat_String	79.71 ns	3.312 ns	9.767 ns	0.0088	56 B

SearchValues

Optimized in scenarios where the **same set of values** is frequently used for searching at runtime.

Currently, only **byte** and **char** supported

“String list” will be overloaded in .**NET 9**

```
public class SearchValueBenchmark
{
    private readonly static char[] vowels = ['a', 'e', 'ı', 'i', 'o', 'ö', 'u', 'ü'];
    private readonly SearchValues<char> searchValues = SearchValues.Create(vowels);

    [Benchmark]
    [Arguments("Hello World!")]
    [Arguments("Lorem ipsum dolor sit amet")]
    0 references
    public int IndexOfCharBenchmark(string text)
    {
        return text.AsSpan().IndexOfAny(vowels);
    }

    [Benchmark]
    [Arguments("Hello World!")]
    [Arguments("Lorem ipsum dolor sit amet")]
    0 references
    public int IndexOfSearchValuesBenchmark(string text)
    {
        return text.AsSpan().IndexOfAny(searchValues);
    }
}
```

Method	text	Mean
IndexOfCharBenchmark	Hello World!	35.815 ns
IndexOfSearchValuesBenchmark	Hello World!	9.948 ns
IndexOfCharBenchmark	Lorem(...) amet [26]	49.738 ns
IndexOfSearchValuesBenchmark	Lorem(...) amet [26]	10.217 ns

Benchmark Results of the Improvements

int.ToString()

Method	Job	Runtime	i	Mean	Ratio	Allocated	Alloc Ratio
Int32ToString	.NET 7	.NET 7.0	12	15.917 ns	1.00	32 B	1.00
Int32ToString	.NET 8	.NET 8.0	12	4.298 ns	0.29	-	0.00
Int32ToString	.NET 7	.NET 7.0	123	19.008 ns	1.00	32 B	1.00
Int32ToString	.NET 8	.NET 8.0	123	3.198 ns	0.18	-	0.00
Int32ToString	.NET 7	.NET 7.0	1234567890	28.095 ns	1.00	48 B	1.00
Int32ToString	.NET 8	.NET 8.0	1234567890	20.994 ns	0.77	48 B	1.00

```
public class ConvertBenchmark
{
    [Benchmark]
    [Arguments(12)]
    [Arguments(123)]
    [Arguments(1_234_567_890)]
    0 references
    public string Int32ToString(int i) => i.ToString();
}
```

Benchmark Results of the Improvements

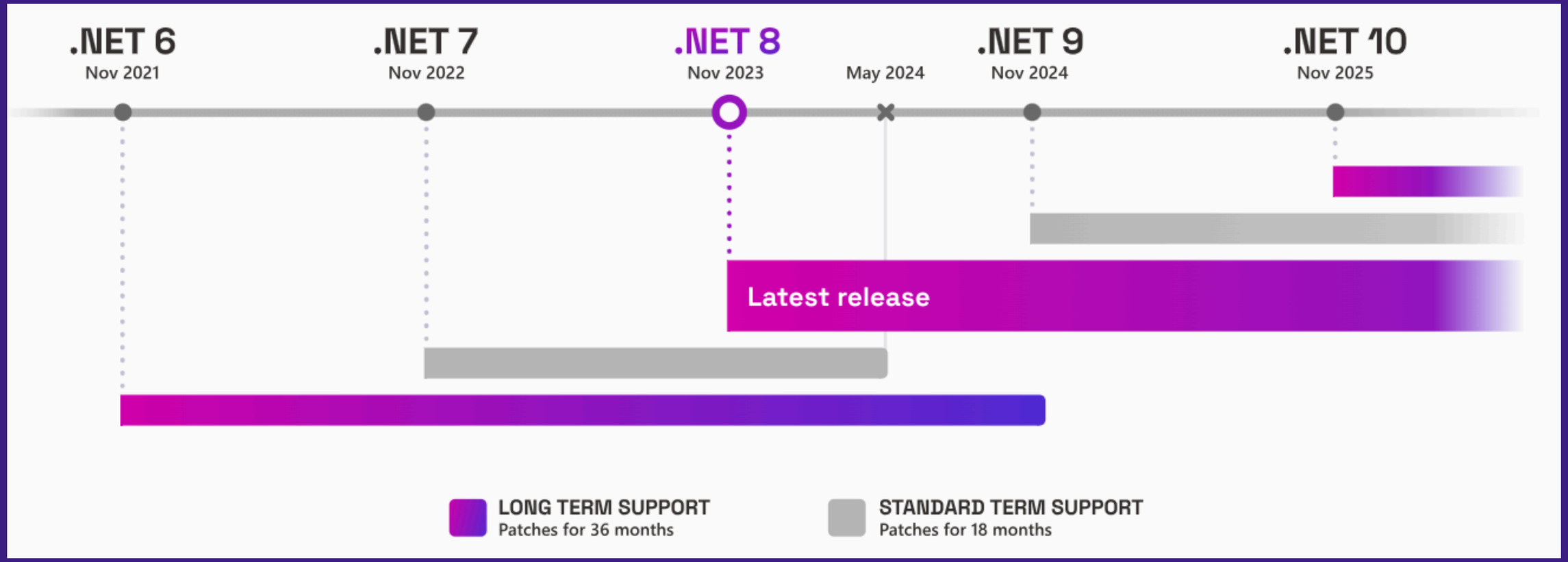
string.Replace()

Method	Job	Runtime	Mean	Ratio
Replace	.NET 7	.NET 7.0	3,573.9 ns	1.00
Replace	.NET 8	.NET 8.0	458.5 ns	0.13

```
public class ReplaceBenchmark
{
    private readonly StringBuilder _sb = new StringBuilder()
        .Append("Shall I compare thee to a summer's day? ")
        .Append("Thou art more lovely and more temperate: ")
        .Append("Rough winds do shake the darling buds of May, ")
        .Append("And summer's lease hath all too short a date; ")
        .Append("Sometime too hot the eye of heaven shines, ")
        .Append("And often is his gold complexion dimm'd; ");

    [Benchmark]
    0 references
    public void Replace()
    {
        _sb.Replace("summer", "winter");
        _sb.Replace("winter", "summer");
    }
}
```

.NET Release Cadence



Thanks for listening

QR to slide

QR to reach me