C# 6.0 und die .NET Compiler Platform (Roslyn)

Hans Peter Bornhauser

Partner: Microsoft Computerworld

DIGICOMP

Worum geht es?

- Neue Sprachfeatures in C# 6.0
- Neues in Visual Studio 2015
- Roslyn – Compilerplattform
- Ausblick auf nächste Version

Referent: Hans Peter Bornhauser

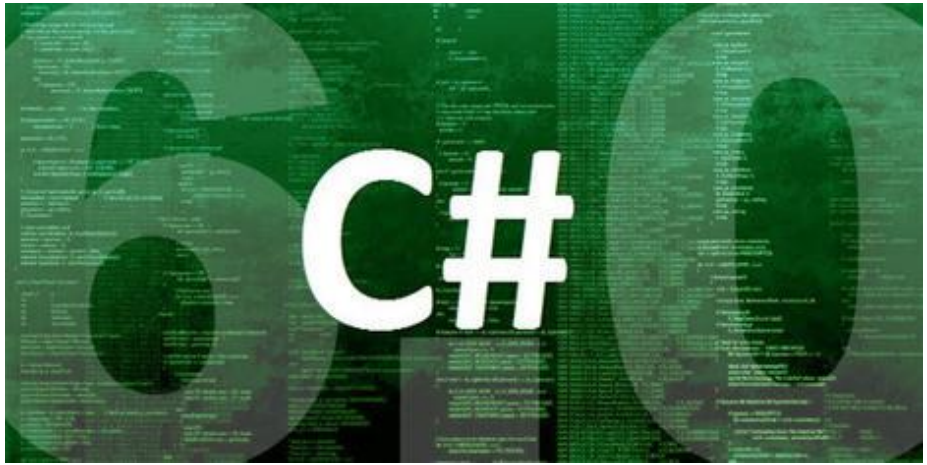
- Dipl. Ing ETH
- Software Architekt, Senior Software Engineer, Berater und Trainer bei Noser Engineering AG in Winterthur
- .NET Trainer bei Digicomp
- 14 Jahre Erfahrung in.NET Entwicklung(seit Beta 1.0)
  - ASP.NET MVC
  - WPF und Silverlight
  - WCF und Entity Framework
- Microsoft Certifications
  - MCSD Web Applications .NET 4.5
  - MCT (Certified Trainer)

NOSER ENGINEERING
WE KNOW HOW

Neues in C# 6.0

- Initialisierung von Auto Properties
- Auto Properties ohne Setter
- Expression Bodies
- Verwendung statischer Klassen
- Der «Null-conditional» Operator
- String Interpolation
- nameof
- Intializer für Collections mit Indexzugriff
- Exception Filter
- await in catch- und finally-Blöcken

C# 6.0 Demo



DIGICOMP

Initialisierung von Auto Properties

- Auto Properties können direkt mit einem Wert initialisiert werden

```
public string Name { get; set; } = "Hejlsberg";
private IList<string> Hobbies { get; set; } = new List<string>();
```

- Dies funktioniert auch ohne Setter

```
public Guid Id { get; } = Guid.NewGuid();
public DateTime CreationDate { get; } = DateTime.Now;
```

- Somit gibt es jetzt echte readonly Properties
  - Getter Properties dürfen im Konstruktor initialisiert werden

Expression Bodies

- Bisher

```
public Func<int, int, int> Add
{ get { return (x, y) => x + y; } }
```

- Neu

```
public int Add(int x, int y) => x + y;
```

- Weitere Beispiele

```
public double Distance(int x, int y) => Math.Sqrt(x * x + y * y);
public double Volume(double a, double b, double c) => a * b * c;
```

- Einschränkungen
  - Keine geschwungenen Klammern im Ausdruck, also nur Rückgabewert

Verwendung statischer Klassen

```
using static System.Math;
using static System.Console;
using static System.Linq.Enumerable;

public static void Main(string[] args)
{
   double degrees = 90;
   double angle    = PI * degrees / 180.0; // Math.PI
   double sinAngle = Sin(angle);           // Math.Sin
   WriteLine("angle {0}", sinAngle);       // Console.WriteLine
   var range = Range(1, 10);               // Enumerable.Range
   var odd = Where(range, i => i % 2 == 1);// Extension Method: Error
   var oddNumbers = range.Where(i => i % 2 == 1);
}
```

- Sinnvoll für mathematische Funktionen und tw. Konstanten

Der «Null-conditional» Operator

- Keine expliziten null-Checks mehr
- Anstatt

```
string name = null;
var customer = GetCustomer();
if (customer != null)
   name = customer.Name;
```

- schreibt man

```
string name = GetCustomer()?.Name;
```

- oder

```
int? length = customers?.Length;
```

- ohne Nullable Type

```
int length = customers?.Length ?? 0;
```

«Null-conditional» Operator als Event Handler

```
public event PropertyChangedEventHandler PropertyChanged;
protected virtual void OnPropertyChanged(string propertyName)
{
    // until C# 5.0
    if (PropertyChanged != null)
    {
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
    }

    // without null check, thread-safe
    PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
```

String Interpolation

- Keine nummerierten Platzhalter mehr

```
var s = $"{p.Name,20} is {p.Age:D3} year{{s}} old";
// kompiliert zu
// string.Format("{0,20} is {1:D3} year{{s}} old", p.Name, p.Age);
```

nameof

- Namen von Variablen zu Compilezeit festlegen (=> Refactoring)

```
throw new ArgumentNullException(nameof(parameter));
public string FirstName
{
   get { return _firstName; }
   set
   {
      if (String.IsNullOrEmpty(value))
         throw new ArgumentException(nameof(FirstName)); // Verwendung von nameof
      _firstName = value;
      OnPropertyChanged(); // C# 5 mit CallerMembername
      OnPropertyChanged(nameof(FullName)); // Fullname ändert wenn Firstname ändert
   }
}

private void OnPropertyChanged([CallerMemberName] string propertyName = "")
{
   PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
}
```

Intializer für Collections mit Indexzugriff

- Syntactic Sugar

```
var coordinate = new JObject
{
    ["x"] = 3,
    ["y"] = 7
};
var testValues = new Dictionary<int, string>
{
    [2] = "V2",
    [5] = "V5"
};
```

- Unterscheidet sich kaum von

```
var coordinate = new JObject
{
    { "x", 3 },
    { "y", 7 }
};
```

Exception Filters

```
try
{
   var content = File.ReadLines(fileName);
}
catch (ArgumentException) when (fileName == string.Empty)
{
   Console.WriteLine("filename is empty");
}
catch (ArgumentException ex) when (ex.Message.Contains("illegal"))
{
   Console.WriteLine("invalid characters in filename");
}
catch (FileNotFoundException)
{
   Console.WriteLine($"file '{fileName}' not found");
}
catch (Exception ex)
{
   Console.WriteLine("Unknown error: " + ex.Message);
}
```

await in catch- und finally-Blöcken

- await ist auch in catch und finally Blöcken erlaubt

```
ServiceClient service = new ServiceClient();
string data = "C# is great";

try
{
    await service.SaveAsync(data);
}
catch (Exception ex)
{
    await LoggingService.LogAsync(ex); // neu: await in catch Block
    throw;
}
finally
{
    await service.CloseAsync(); // neu: await in finally Block
}
```

Roslyn – die .NET Compiler Platform

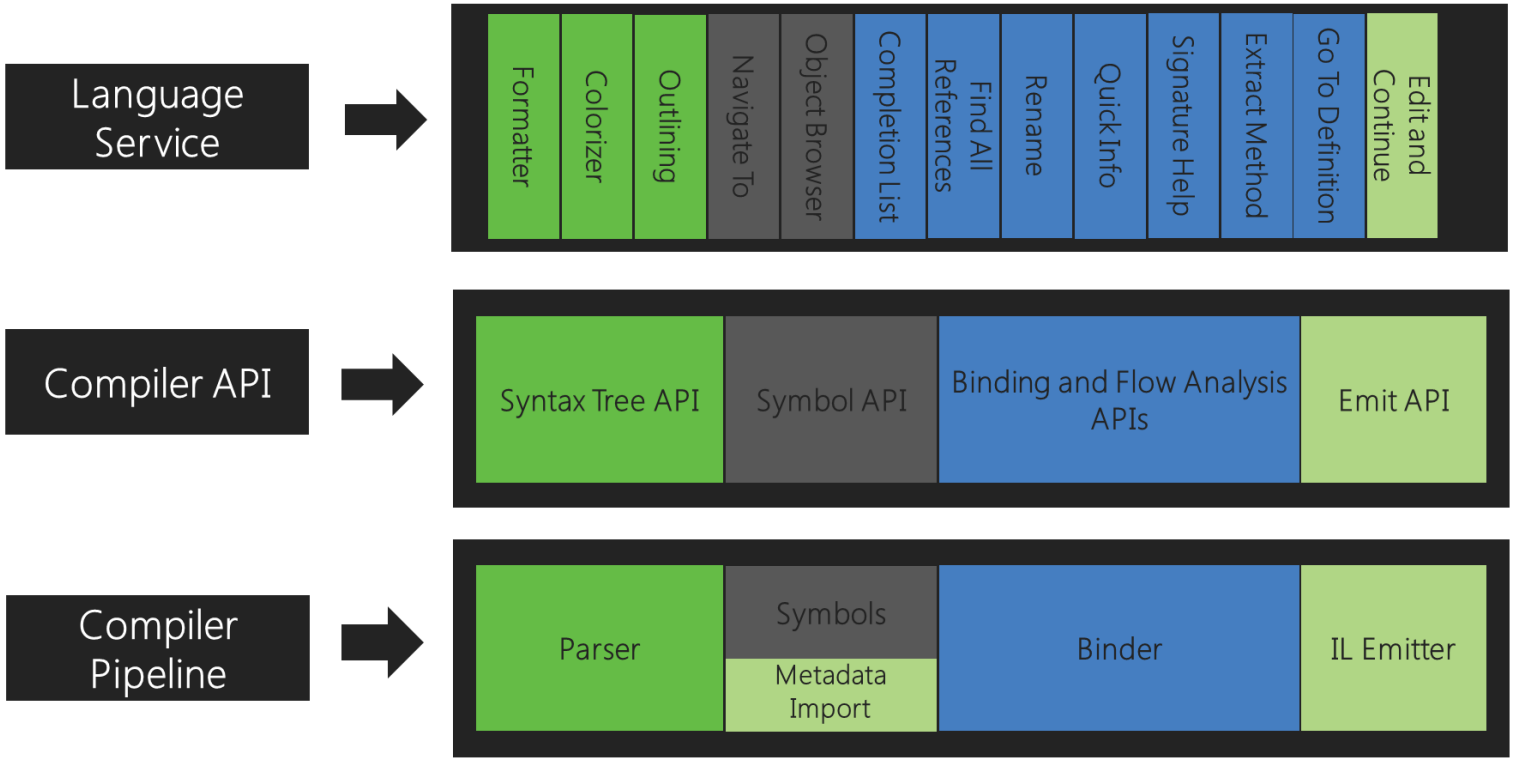
Was ist Roslyn?

- Ein komplett neuer Compiler (C# und VB.NET)
  - Modular
  - Open Source
- Diverse APIs für Compiler und Workspaces
- Geeignet für komplett neue Tools, die auf dem Compiler aufsetzen
  - Domain specific Languages
  - Neue Tools für Compiler (analog Resharper)

Warum ein neuer Compiler?

- Die alten Compiler waren in C++ geschrieben
  - «Eat your own dogfood» Mantra
- Mehrere Compiler notwendig
  - Batch Compiler (csc.exe, vbc.exe)
  - Hintergrund-Compiler in Visual Studio (für Editor)
  - Snippet-Compiler für Immediate Window
- Neue Anforderungen
  - Experimente mit Sprachen
  - Öffnung des API für 3rd party

Compiler Pipeline



Syntax Tokens und Trivia

- Parser generiert Tokens und Trivia
- Tokens sind Sprachelemente
  - Schlüsselwörter
    - static, class, foreach, if, ...
  - Identifier
    - System, Console, WriteLine, x, ...
  - Operatoren: +, -, *, /, %
  - Punktuierung: . , ; ...
- Trivia verändern das Programm nicht, müssen aber erhalten bleiben
  - Kommentare
  - Whitespace: Leerzeichen, \r\n, EOF

Arbeit mit Syntax Tree: Iteration

```
var tree = CSharpSyntaxTree.ParseText("class Foo { void Bar() {} }");
var node = (CompilationUnitSyntax)tree.GetRoot();


foreach (var member in node.Members)
{
    if (member.CSharpKind() == SyntaxKind.ClassDeclaration)
    {
        var @class = (ClassDeclarationSyntax)member;

        foreach (var member2 in @class.Members)
        {
            if (member2.CSharpKind() == SyntaxKind.MethodDeclaration)
            {
                var method = (MethodDeclarationSyntax)member2;
                // do stuff
            }
        }
    }
}
```

Verwendung von LINQ für Abfrage des SyntaxTree

```
var bars = from member in
                    node.Members.OfType<ClassDeclarationSyntax>()
           from member2 in
                    member.Members.OfType<MethodDeclarationSyntax>()
           where member2.Identifier.Text == "Bar"
           select member2;

var result = bars.ToList();
```

Verwendung eines Visitors

```
class MyVisitor : CSharpSyntaxWalker
{
    public override void VisitMethodDeclaration(MethodDeclarationSyntax node)
    {
        if (node.Identifier.Text == "Bar")
        {
            // do stuff
        }

        base.VisitMethodDeclaration(node);
    }
}

new MyVisitor().Visit(node);
```

Analyzer und Code Fixes

- Analyzer analysieren den SyntaxTree
- Code Fixes reparieren bei Bedarf

Roslyn - Demo

Was ist (wirklich) neu in Visual Studio 2015?

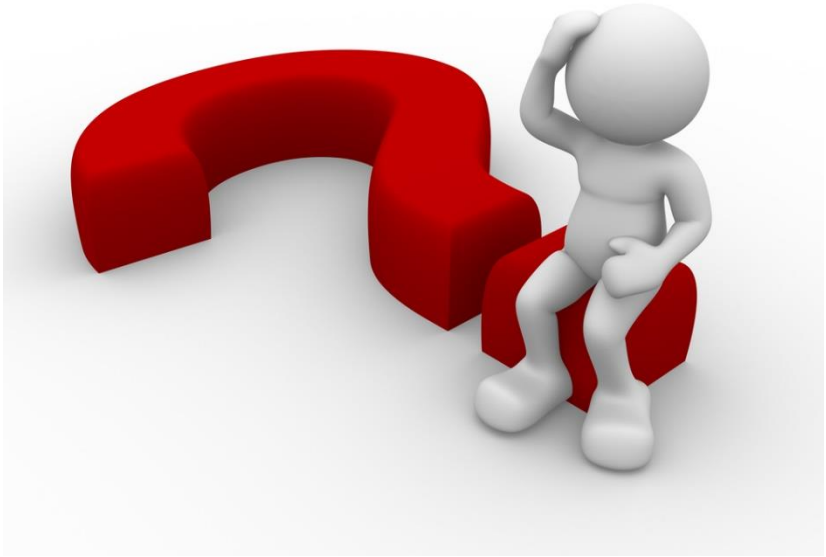
- Der Compiler!
  - Glühlampe: Live Code Analyse
- Lambda Expressions in Watch Variablen
- Cross-Platform Entwicklung und Debugging
  - Apache Cordova
  - Xamarin
  - Android
- Verbesserungen der CodeLens

Was kommt als Nächstes?

(frühe) Ideen für C# 7 (Mads Torgersen)

- Mehr Elemente aus der funktionalen Programmierung
  - Pattern Matching
  - Weniger Objekte kopieren: return ref, Zugriff auf Teile von Arrays, ...
- Neue Syntax für Tuples: `(int x, int y) Method (...) { .... return (1, 3); }`
- Sprachfeatures für State Machines
- Null Referenzen verhindern durch Compiler Prüfungen
  - Absicht bekannt geben: `public string ? LastName; // value could be null`

F&A



Weiterführende Kurse

- Neues in .NET 4.6 und Visual Studio 2015 (CN6)
  - 1 Tag
  - CHF 800.–
  - Nächster Termin: 17.11.2015
  - https://www.digicomp.ch/weiterbildung/softwareentwicklung/microsoft-net/microsoft-net-framework/neues-in-net-4-6-und-visual-studio-2015
- Programmieren mit C# (CSW)
  - 3-5 Tage
  - CHF 3’250.–
  - https://www.digicomp.ch/weiterbildung/softwareentwicklung/microsoft-net/c/programmieren-mit-c-net-framework-4-6

Referenzen

- http://www.github.com/dotnet/roslyn
- https://channel9.msdn.com/Blogs/Seth-Juarez/Looking-Ahead-to-C-7-with-Mads-Torgersen