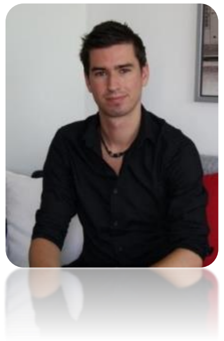
„Back to the Future“

Eine Zeitreise von C# 1.0 zu C# 7.0

About us

- David Tielke
- www.David-Tielke.de
- mail@david-tielke.de
- Twitter: @davidtielke

- Ing. Christian Giesswein, MSc.
- www.software.tirol
- christian@software.tirol
- Twitter: @giessweinapps



Sprachversionen von C#

- C# 1.0 (2002) – Baseline
- C# 1.2 (2003) – ECMA Anpassungen
- C# 2.0 (2005) – Generics
- C# 3.0 (2007) – LINQ
- C# 4.0 (2010) – Dynamische Bindung
- C# 5.0 (2012) – Async & await
- C# 6.0 (2015) – String Interpolation, Null Propagation
- C# 7.0 (Future)

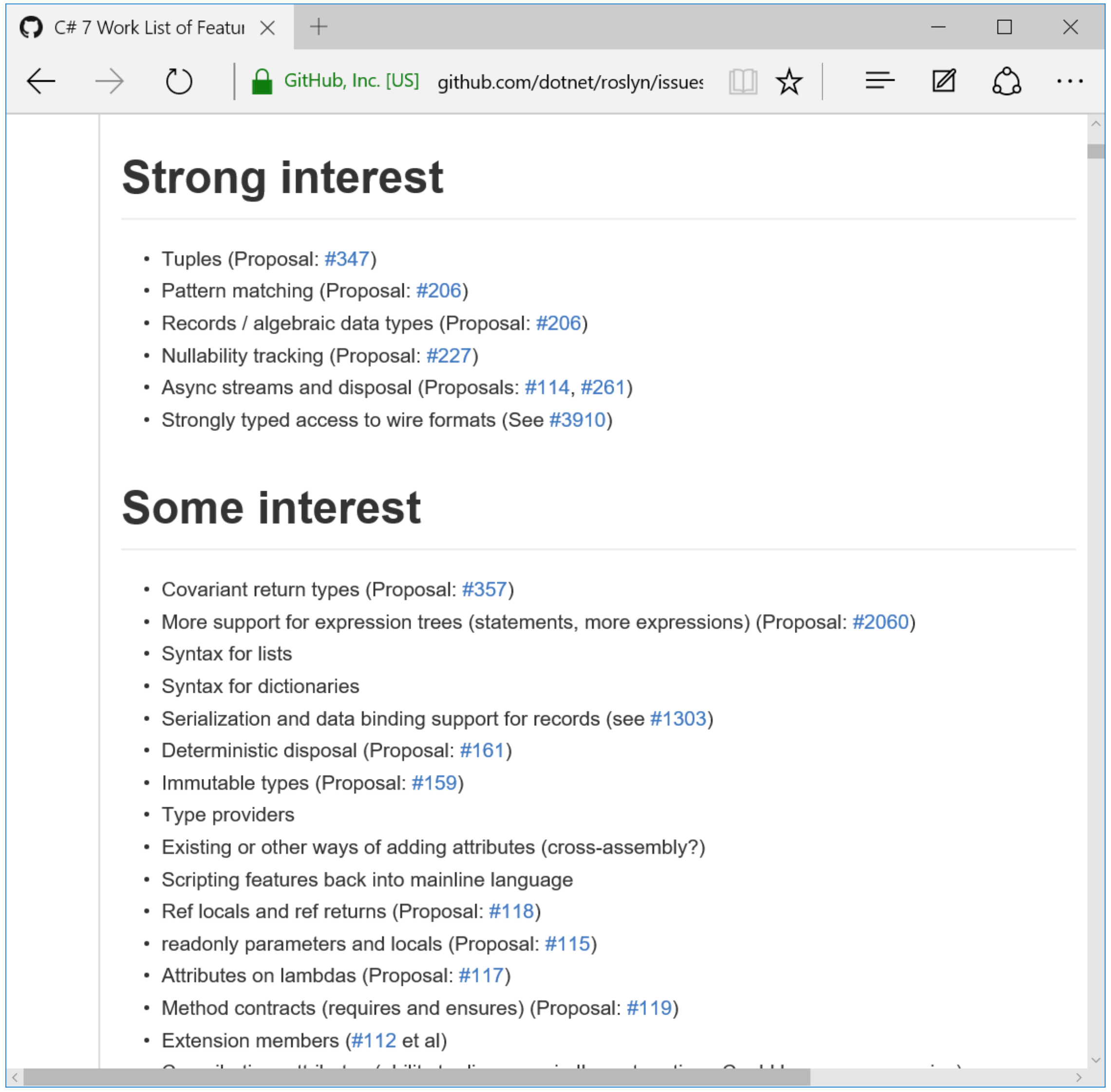
C#

- If-Schleife

C# vNext (7.0)

Stand C# 7.0

- Komplett transparent auf github
- Releasedatum: Unklar
- Umfang: Unklar²
- Syntax: Unklar³



Strong interest

- Tuples (Proposal: #347)
- Pattern matching (Proposal: #206)
- Records / algebraic data types (Proposal: #206)
- Nullability tracking (Proposal: #227)
- Async streams and disposal (Proposals: #114, #261)
- Strongly typed access to wire formats (See #3910)

Some interest

- Covariant return types (Proposal: #357)
- More support for expression trees (statements, more expressions) (Proposal: #2060)
- Syntax for lists
- Syntax for dictionaries
- Serialization and data binding support for records (see #1303)
- Deterministic disposal (Proposal: #161)
- Immutable types (Proposal: #159)
- Type providers
- Existing or other ways of adding attributes (cross-assembly?)
- Scripting features back into mainline language
- Ref locals and ref returns (Proposal: #118)
- readonly parameters and locals (Proposal: #115)
- Attributes on lambdas (Proposal: #117)
- Method contracts (requires and ensures) (Proposal: #119)
- Extension members (#112 et al)

Mehrere Rückgabewerte

- out- oder ref-Parameter
- Tuple<T1,T2>
- Eigener Datentyp

Tupel

```
public (int sum, int count) Foo(…)
{
   int sum = 1;
   int count = 2;
   return (sum, count);
}
```

Tupel - Literale

```
public (int sum, int count) Foo(…)
{
   return new (int sum, int count) { sum = 1, count = 2 };
}
```

Tupel - Deconstruction

```
public (int sum, int count) Foo(...) { ... }
public void Main()
{
   int sum, count = 0;
   (sum, count) = Foo(...);
   (var sum1, var count2) = Foo(...);
   Console.WriteLine($"{sum} - {count} : {sum2} - {count2}");
}
```

Records

```
public class Cartesian(double x: X, double y: Y);
```

Records – In C# 6.0

```
public class Cartesian
{
        private readonly double $X;
        private readonly double $Y;
        public Cartesian(double x, double y) { $X = x; $Y = y; }
        public double X { get { return $X; } }
        public double Y { get { return $Y; } }
        public static bool operator is(Cartesian c, out double x, out double y) { ... }
        public override bool Equals(object obj) { ... }
        public override int GetHashCode() { ... }
        public override string ToString() { ... }
}
```

Code abhängig vom Typ ausführen

```
public class Person(string vorname : Vorname, string nachname : Nachname);

public void Add(object obj)
{
   if (obj is Person)
   {
      Person p = (Person)obj;
      Console.WriteLine(p.Vorname);
   }
}
```

Pattern Matching (1)

```
public class Person(string vorname : Vorname, string nachname : Nachname);

public void Add(object obj)
{
   if (obj is Person p)
   {
      Console.WriteLine(p.Vorname);
   }
}
```

Pattern Matching (2)

```
public class Person(string vorname : Vorname, string nachname : Nachname);

public string Foo(object obj)
{
   switch(obj)
   {
      case Person(var vn, "Tielke"): return $"{vn} gehört zu meiner Familie";
      case Person(var vn, var nn): return $"{vn} gehört zur Familie {nn}";
   }
}
```

Nullability Checking – General References

```
public void Foo()
{
   Person p = new Person(...);
   Console.WriteLine(p.Vorname);
}
```

Nullability Checking – Nullable References

```
public void Foo()
{
   Person? p = new Person(...);
   Console.WriteLine(p.Vorname); // Compilerfehler
   if(p != null)
   {
      Console.WriteLine(p.Vorname);
   }
}
```

Nullability Checking – Mandatory References

```
public void Foo()
{
   Person! p = new Person(...);
   p = null; // Compilerfehler
   Console.WriteLine(p.Vorname);
}
```

Covariant Return Values

```
class Compilation
{
    virtual Compilation WithOptions(Options options){...}
}

class CSharpCompilation : Compilation
{
    override CSharpCompilation WithOptions(Options options){...}
}
```

Immutable Types

```
public immutable class Person
{
   public Person(string firstName, string lastName, DateTimeOffset birthDay){...}

   public string FirstName { get; }
   public string LastName { get; }
   public DateTime BirthDay { get; }

   public string FullName => $"{FirstName} {LastName}";
   public TimeSpan Age => DateTime.UtcNow - BirthDay;
}
```

Readonly Parameters and Locals

```
public void Foo(readonly int blubb)
{
   readonly bar = 4;
}
```

Method Contracts

```
public int Add(Person personToAdd)
   requires personToAdd != null
   ensures return > 0
{
   // ...
}
```