BASTA!®
.NET, VISUAL STUDIO & MORE

Oliver Sturm, DevExpress

# Was bringt die Zukunft für C#?

# Wer ist eigentlich dieser Oliver Sturm?

- Ich bin
  - Interessiert an Programmiersprachen, Datenbanken und vielen anderen Sachen
  - Microsoft MVP für C#
  - Autor der C# Corner im dot.net magazin
- Ich
  - Arbeite für Developer Express als Technical Evangelist und Lead Program Manager
  - Blogge unter http://www.sturmnet.org/blog
- Sie sollten
  - Mir Emails schicken an oliver@sturmnet.org

# Agenda

- Etwas Geschichte und (neue) Grundlagen
- C# 4.0
  - “dynamic”
  - Benannte und optionale Parameter
  - Office-Automatisierung ohne PIA (Primary Interop Assembly)
  - Co- und Kontravarianz
- Und danach…
  - Compiler as a service – C# Futures

# C# History nach Anders Hejlsberg

- C# 1.0 – erste Version
- C# 2.0 – was in 1.0 nicht mehr reinpasste vor dem Release
- C# 3.0 – LINQ, funktionale Einflüsse
- C# 4.0 – Dynamic Programming
- C# 5.0 …

# Deklarative Ideen

- Was soll passieren, nicht wie soll es passieren
- Geht zusammen mit funktionalen Ansätzen für Parallelisierung
- LINQ und auch “dynamic” können so interpretiert werden, aber so richtig ist sich Anders darüber wohl auch nicht klar

# Dynamik

- Die Welt ist heute dynamisch – das hat auch Anders gemerkt :-)
- Die DLR ist ein wichtiger neuer Bestandteil von .NET
- Dynamik ist ein Bestandteil von deklarativem Programmieren
- C# 4.0 enthält wichtige Bauteile

# C# 4.0 – “dynamic”

- Der Umgang mit statischen .NET-Klassen ist einfach in C#
- Sind die Typen unbekannt, kann Reflection helfen
- Andere “Sprachwelten” haben ihre eigenen Techniken (Python, Ruby, aber auch Automation, Web Services, …)
- “dynamic” bringt diese verschiedenen Ansätze zusammen
- IDynamicObject erlaubt die Implementation dynamischer Typen in C#

# C# 4.0 – “dynamic”

```
dynamic i = 42;
dynamic s = “Hi there”;
```

- Unterscheidung zwischen Design Time- und Runtime-Typen
- “Late Binding”, “Dynamic Dispatch”
- Compiler setzt Syntax um in Objektinstanzen der DLR

# Demo

"dynamic"

# Optionale Parameter

```
void Function(int a = 20, string b = “Text”) { ... }

Function();
Function(99);
```

- Parameter können Standardwerte haben
- Optionale Parameter müssen immer hinter Pflichtparametern stehen
- In IL werden diese Parameter mit Attributen dekoriert => Kompatibilität mit anderen Sprachen

# Benannte Parameter

```
void Function(int a = 20, string b = “Text”) { ... }

Function(b: “Different Text”);
Function(a: GetMyA(), b: GetMyText());
```

- Parameter mit Namen angeben beim Funktionsaufruf
- Ebenfalls Attribute in IL
- Zusammen mit optionalen Parametern sehr mächtig, aber auch großes Potential für Verwirrung

# COM Interop

- Profitiert von “dynamic”, sowie von benannten und optionalen Parametern
- Typen aus PIAs (primary interop assembly) können importiert werden
- Andere Kleinigkeiten, von denen ich wenig Ahnung habe ☺
  - Indizierte Properties

# Demo

Benannte und optionale Parameter (und ein bischen COM mit Excel)

# (Co-) Varianz

- Ein Vogel ist ein Tier – check
- Eine Gruppe Vögel ist eine Gruppe Tiere – ch… hm, nicht in C# 3.0
- Wir erwarten, dass ein Vogel auch als Tier zu betrachten ist, unter allen Umständen, und in C# 4.0 ist das auch (fast) so

- Varianz sorgt dafür, dass alles funktioniert, wie man denkt

# Varianz, die Theorie

- Zwei Typen T und U können folgende Verhältnisse haben:
  - T ist kleiner als U
  - T ist größer als U
  - T ist gleich U
  - T und U sind unabhängig
- Eine Operation, die mit T und U arbeitet und deren Verhältnis beibehält, nennt man covariant
- Eine Operation, die “kleiner” und “größer” umkehrt, nennt man kontravariant



Explanation stolen from Eric Lippert without any permission whatsoever

# Co- und Kontravarianz mit Delegates (C# 2.0)

```
Bird CreateBird() { ... }
Func<Animal> function = CreateBird;
```

```
void TakeTiger(Tiger tiger) { ... }
void TakeAnimal(Animal animal) { ... }

Action<Mammal> action1 = TakeTiger; // nicht erlaubt
Action<Mammal> action2 = TakeAnimal; // erlaubt
```

- Func<Animal> is… …als Func<Bird>

Nicht mö… …”

Funktioniert. Da Animal > Mammal, aber Action<Mammal> > Action<Animal>, ist dieser Vorgang kontravariant.



Example also stolen from Eric Lippert, he's really gonna be pissed if he finds out

# Varianz: neu in C# 4.0

```
List<string> strings = new List<string>{“one”, “two”};
IEnumerable<object> objects = strings;
```

- Das war früher nicht möglich, da objects.Add(42); zu verhindern war
- In C# 4.0 über eine Erweiterung der generischen Typparameter möglich (“out” für Covarianz, “in” für Kontravarianz)
- Die Erweiterungen funktionieren nur für Interfaces und Delegates
- Leider sind die geänderten Typen noch nicht in der VS 2010 CTP

# Demo

Varianz in C# 4.0

# Was danach kommt – Compiler as a Service

- Neuer C#-Compiler, geschrieben in C# - schon seit 16 Monaten in Arbeit
- Implementiert als wiederverwendbares Modul
- Verwendung des Compilers aus eigenen Programmen heraus wird einfacher
- Vorteile
  - Eval
  - Embedding, DSLs, …
  - Meta-programming (etwas unklar)
  - Language Object Model

# Zusammenfassung

- Dynamik ist das große Thema in C# 4.0
- Stoßrichtung deklarative Programmierung
- Modularisierung für die weitere Zukunft
- Ebenfalls Zukunft: Erweiterungen am Expression-System, Sprachunterstützung

# Vielen Dank

Bitte kontaktieren Sie mich jederzeit zum Inhalt dieser Session

oliver@sturmnet.org