entwickler.press

F#

Einstieg und praktische Anwendung

Oliver Sturm

Oliver Sturm

F#

Einstieg und praktische Anwendung

entwickler.press

Oliver Sturm
F#

ISBN: 978-3-86802-270-4

Ein Imprint der Software & Support Media GmbH

Bibliografische Information Der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.ddb.de abrufbar.

Ihr Kontakt zum Verlag und Lektorat:
Software & Support Media GmbH
entwickler.press
Darmstädter Landstr. 108
60598 Frankfurt am Main
Tel.: +49 (0)69 630089-0
Fax: +49 (0)69 930089-89
lektorat@entwickler-press.de
http://www.entwickler-press.de

Lektorat: Sebastian Burkart
Korrektorat: Frauke Pesch
Satz: Laura Acker

Inhaltsverzeichnis

Vorwort

F# ist eine der neuesten Programmiersprachen der .NET-Plattform, besonders unter denjenigen Sprachen, die von Microsoft selbst zusammen mit Visual Studio an Entwickler ausgeliefert werden. Allerdings ist die Sprache schon seit langer Zeit in der Entwicklung – bereits in 2005 kam die Version 1.0 von F# heraus – und sie basiert auf Ideen und sogar auf einer Syntax, die von anderen Sprachen und Plattformen übernommen wurden und schon lange vor der Zielplattform .NET existierten.

Ein paar Jahre ist es her, dass Microsoft über das Blog von „Soma" Somasegar offiziell ankündigte, dass F# mit Visual Studio 2010 ausgeliefert werden würde. Das führte zunächst zu einiger Hektik im F#-Team, da noch vieles perfektioniert werden musste, das bisher für ein per separatem Download verfügbares F# nicht für sehr wichtig gehalten worden war: ein aktualisiertes Projektsystem inklusive einer ordentlichen Visual-Studio-Anbindung, IntelliSense auf einem aktuellen Stand der Technik, ein paar neue Projektvorlagen und einiges mehr.

Schon seit frühen Zeiten seiner Entwicklung war F# bei bestimmten Anwendergruppen auf reges Interesse gestoßen. Als eine Sprache, die sich funktionale Einflüsse zumindest als einen wichtigen Punkt auf die Fahne schrieb, bot sich F# offensichtlich für diejenigen algorithmisch geprägten Projekte an, die schon immer gern funktional geführt wurden. Aber mit dem Hintergrund der breit gefächerten Funktionalität des .NET Frameworks gab es hier etwas, das keine andere funktionale Sprache bieten konnte: die Fähigkeit, Anwendungen für die verschiedensten Zielsysteme zu schreiben, mit und ohne grafische Benutzerschnittstelle, Server oder Client, mit einfachstem Datenbankinterface,

als Webanwendung oder für Mobilplattformen. Gleichzeitig ist F# selbst hybrid in seinen Ansätzen, sodass die Interaktion mit all diesen verschiedenen Subsystemen so einfach ist, wie Sie sich nur vorstellen können.

Seither hat sich die Entwicklung weiter fortgesetzt, sowohl im Bereich der Sprachen als auch für .NET insgesamt. Für F# bedeutet das hauptsächlich, dass seine Ideen heute relevanter sind als je zuvor. In C# und VB.NET haben seit Visual Studio 2008 immer mehr funktionale Ideen Einzug gehalten, in großem Umfang mit der Sprachunterstützung für das sehr funktional ausgeprägte LINQ, das in .NET 3.5 eingeführt wurde. Parallelisierbarkeit ist ein Thema, das aufgrund der typischen Architekturen aktueller Prozessoren fast jede Software betrifft und über das viele Programmierer in die Welt der funktionalen Programmierung finden, da sie sich von puren Funktionen und unveränderbaren Daten neue Lösungswege versprechen. F# bietet als Programmiersprache den besten Weg zu diesen funktionalen Zielen, der Ihnen auf der .NET-Plattform offensteht, während die Mittel der Objektorientierung, mit denen Sie vertraut sind, nicht zurückstehen müssen.

Die Vergangenheit hat gezeigt, dass F# eine Plattform der Entwicklung in mehr als einer Hinsicht ist: Sprachfeatures aus F# haben oft den Weg in andere .NET-Sprachen gefunden und maßgeblich dazu beigetragen, dass wichtige Neuerungen in die Plattform integriert wurden. Ob Sie dabei an große Dinge denken, wie Generics in .NET 2.0, oder an kleinere, wie die Einführung der Tupel-Typen in .NET 4.0 – F# spielt in diesen Bereichen immer wieder eine Vorreiterrolle. Das wird sich auch in Zukunft nicht ändern, denn die in C# 5.0 neuen Async-Schemata gibt es in analoger Form in F# schon lange, und die für F# 3.0 angekündigte Funktionalität zu Information Rich Programming verspricht wiederum neue Fähigkeiten, die womöglich in Zukunft in Richtung C# oder VB.NET migriert werden könnten.

Vielleicht sind Sie ein Programmierer, dessen Arbeit sich um die Pflege vorhandener Software dreht. In diesem Fall sind Sie ständig auf der Suche nach dem einfachsten Weg, Ihre Software auf einem modernen Stand zu halten und sich von der Konkurrenz abzusetzen. Die Pflegbarkeit Ihres Codes ist für Sie wichtig, da dessen Langlebigkeit für Sie bereits alltäglich ist. Es ist gut möglich, dass F# Ihnen wichtige Vorteile verschaffen kann: Sie können Algorithmen einfacher, übersichtlicher und kompakter implementieren, während die moderne und fortschreitende Entwicklung der Sprache Ihnen hilft, Alleinstellungsmerkmale zu erarbeiten. Durch die Kompatibilität der Sprache mit .NET, und so mit allen anderen Sprachen der Plattform, können Sie einen Umschwung graduell einleiten oder auch nur ausnahmsweise F# einsetzen, wenn es gerade passt.

Oder sind Sie ein Programmierer, der nach der richtigen Sprache für eine neue .NET-Entwicklung sucht? Dann sind Sie bereits vertraut mit der Funktionalität, die Sie in der Plattform finden. Um im Markt erfolgreich zu sein, müssen Sie den effizientesten Weg finden, die Plattform zu nutzen. Je schneller und einfacher Sie Code schreiben können, desto besser für Ihr Geschäft. F# sollte Ihnen gut gefallen, denn Code in der Sprache ist typischerweise kürzer als der, den Sie in C# oder VB.NET schreiben können – natürlich bei gleicher Funktionsweise! Sie bekommen aber auch ein breiteres Spektrum an Programmieransätzen geboten, aus denen Sie jeweils den besten für ein Problem auswählen können. Für viele Zwecke lässt sich auch ein funktional purer Ansatz verfolgen, der durch einfache Testbarkeit, Parallelisierbarkeit und die daraus resultierende Stabilität besticht.

Vielleicht sind Sie ein Programmierer, dessen Erfahrungen wesentlich von anderen Plattformen her stammen und für Sie ist .NET neu. Sie mögen in diesem Fall bereits mit objektorientierten oder funktionalen Ideen vertraut sein oder mit beiden. F# sollte in jedem Fall die Sprache Ihrer Wahl sein, denn sie bietet volle Kompatibilität mit .NET bei einfachstem Einstieg, unabhängig von Ihrer bisherigen Ausrichtung.

Weiterentwicklung ist kein Selbstzweck. Ein wichtiger Punkt im Zusammenhang mit F# ist allerdings, dass die Sprache unterschiedliche Ideen vereint, die sich im Laufe vieler Jahre als gleichermaßen legitim erwiesen haben. Der Programmierer mit dem weitesten Horizont gewinnt das Rennen um den Job, den Auftrag, das Geschäft – ich wünsche Ihnen, dass Sie das sind!

1 Hallo F# – Ein praktischer Einstieg

In diesem ersten Kapitel geht es ganz praktisch zur Sache. Menschen lernen auf unterschiedliche Art, und für manchen Programmierer ist der Umgang mit existierendem Code, vielleicht sogar mit vollständigen fertigen Lösungen, der beste Weg, eine neue Technologie zu lernen. Jeder Programmierer schätzt jedoch den ersten Eindruck, den eine kurze Anleitung Schritt für Schritt erzeugen kann. Wenn sich später die Details häufen, ist manches nicht mehr schnell im Kopf wiederzufinden, aber Sie erinnern sich am leichtesten daran, schon einmal bei einer bestimmten Problemlösung zugesehen zu haben.

Darum soll es in diesem Kapitel gehen. Das Problem, das es zu lösen gilt, werden Sie noch im Detail kennen lernen. Die Plattform ASP.NET MVC ist (abgesehen von F# natürlich) das Mittel der Wahl, und auch dazu wird alles Wichtige erklärt.

Es muss klar gesagt sein, dass besonders die gewählte Zielplattform geradezu willkürlich ausgesucht wurde. F# ist eine Programmiersprache, mit der die Plattform .NET programmiert werden kann. Es gibt sicherlich Ziele, die sich besonders einfach oder schwer erreichen lassen, wie das in jeder Sprache der Fall ist. F# als hybride Sprache bietet allerdings wesentlich mehr einfache als schwierige Lösungen. ASP.NET MVC ist also Stellvertreter für viele andere mögliche Systeme der Plattform .NET, die Sie ebenso ansteuern könnten.

Das wichtigste Ziel auch in diesem ersten Kapitel ist es natürlich, den nötigen Fokus auf die Sprache F# zu legen. Damit wird es also losgehen: mit den Grundlagen der Sprache!

1.1 Erste Schritte

Die folgenden kleinen und einfachen Beispiele sollen Ihnen helfen, einen ersten Einblick in die Sprache F# zu gewinnen.

1.1.1 Hallo Welt an der Konsole

Im ersten Beispiel finden Sie eine Implementierung des wohl ältesten Programmierbeispiels der Welt: des Hallo-Welt-Programms. Ein kleiner Zusatz ist auch enthalten, indem das Programm den Anwender nach seinem Wohlbefinden fragt und eine entsprechende Ausgabe erzeugt. Schauen Sie sich diesen Quelltext an:

```
printfn "Hello world!"
printfn "How are you doing?"
printf " -> "

let response = System.Console.ReadLine()

printfn "Great to you hear you're %s!" response
```

Listing 1.1: Hallo Welt

Das übliche Hallo Welt lässt sich natürlich bereits mit der ersten Zeile dieses Beispiels umsetzen. Dort wird lediglich eine eingebaute Funktion *printfn* verwendet, die einen formatierten Text auf der Konsole ausgeben kann. Danach werden einige weitere Ausgaben erzeugt und ein Text wird mithilfe der Standard-.NET-Funktion *ReadLine* abgefragt. In der letzten Zeile können Sie auch den Aspekt der Formatierung erkennen, die von *printfn* unterstützt wird: Der Platzhalter *%s* wird bei der Ausgabe durch den zuvor eingegebenen Text ersetzt.

Nach einem Programmlauf könnte die gesamte Ausgabe wie folgt aussehen:

```
Hello world!
How are you doing?
 -> very well
Great to you hear you‘re very well!
```

Bereits an diesem einfachen Beispiel lassen sich gewisse Aspekte der Sprache F# erkennen. Zunächst fällt auf, dass das Programm einen linearen, skriptartigen Charakter hat. Es gibt keine syntaktischen Konstrukte, wie etwa eine *Main*-Methode oder eine bestimmte Klasse, die zur Ausführung unbedingt erforderlich wären. Auch die Syntax der einzelnen Aufrufe an *printfn* ist sehr einfach, da sie ohne Klammern oder Kommata in der Parameterliste auskommt. Die „Variable“ *response* hat anscheinend keinen bestimmten Typ – tatsächlich hat sie einen Typ, aber dieser wird vom Compiler automatisch hergeleitet. Schließlich ist noch zu sehen, dass der Übergang zum Laufzeitsystem von .NET fließend ist, da zum Einlesen der Benutzereingabe ein simpler Aufruf an eine .NET-Klasse ausgeführt wird. Dies alles sind wertvolle erste Ausdrücke. In den folgenden Kapiteln werden Sie alle notwendigen Details zu diesen Themen erfahren.

1.1.2 Hallo Welt mit Windows Forms

Das zweite Einstiegsbeispiel ist etwas komplizierter als das vorherige. Das folgende F#-Programm verwendet Windows Forms, um ein Fenster mit einem Daten-Grid und einigen Testdaten anzuzeigen. Hier ist der Quelltext:

```
open System.Windows.Forms

type Person = { Name: string; Age: int }
let testData =
  [|
    { Name = "Harry"; Age = 37 };
    { Name = "Susan"; Age = 28 };
    { Name = "Pete"; Age = 43 };
    { Name = "Julie"; Age = 36 }
  |]

let form = new Form(Text = "F# Windows Forms")
let dataGrid =
  new DataGridView(Dock = DockStyle.Fill,
                   DataSource = testData)
form.Controls.Add(dataGrid)

Application.Run(form)
```

Listing 1.2: Hallo Welt mit Windows Forms

In der ersten Zeile wird der Namespace *System.Windows.Forms* für die Verwendung im weiteren Code verfügbar gemacht. Sie kennen diese Technik vielleicht von anderen .NET-Sprachen. Als Folge dieser Deklaration können weiter unten im Code z. B. die Typen *Form* und *DataGridView* verwendet werden, ohne dass ihnen der Namespace vorangestellt werden muss.

Die Zeile, die mit dem Schlüsselwort *type* beginnt, definiert einen Datentyp, genau genommen einen *Record*-Typ. In der „Variablen" *testData* wird ein Array von Objekten erzeugt, die jeweils vom soeben dekla-

rierten Typ *Person* sind. Diese Liste von Testdaten kann unmittelbar als Datenquelle des Daten-Grids verwendet werden, das instanziiert und in der „Variablen" *dataGrid* abgelegt wird. Auch ein Formular wird erzeugt und mit dem Control gefüllt, bevor schließlich ein Aufruf in die *Application*-Klasse von Windows Forms das Programm startet und das Fenster sichtbar macht.

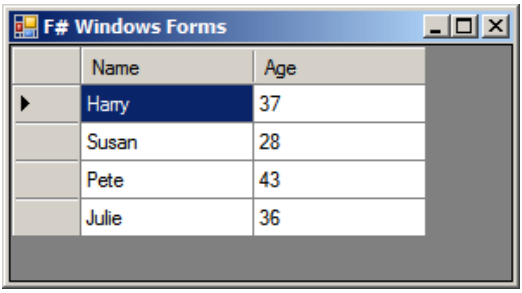

Abbildung 1.1: Hallo Welt als Windows-Forms-Programm

1.1.3 Hallo Welt mit WPF

Das dritte Beispiel dieses Kapitels verwendet Windows Presentation Foundation, kurz WPF, als weitere UI-Plattform zur Erzeugung von Windows-Anwendungen mit F#. Zunächst enthält das Projekt eine XAML-Datei mit Code für das Hauptformular:

```
<Window
  xmlns=
   "http://schemas.microsoft.com/winfx/2006/xaml/
                                              presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="MainWindow" Height="300" Width="500">
```

```
  <Grid>
    <Button Name="clickButton" Content="Click me!"
      Height="40" Width="150" />
  </Grid>
</Window>
```

Listing 1.3: Der XAML-Code für das WPF-Hauptfenster

Der folgende Quelltext stellt das eigentliche Programm dar, mit dem das Hauptformular zur Anzeige gebracht wird:

```
open System
open System.Windows
open System.Windows.Controls

let loadWindow() =
  let resourceLocator =
    new Uri("/HelloWorldWPF;component/MainWindow.xaml",
            UriKind.Relative)
  let window =
    Application.LoadComponent(resourceLocator) :?> Window
  (window.FindName("clickButton") :?> Button).Click.Add(
    fun _ -> MessageBox.Show("Hello World!") |> ignore)
  window

[<STAThread>]
(new Application()).Run(loadWindow()) |> ignore
```

Listing 1.4: Der F#-Code für das WPF-Beispiel

Es sind einige Elemente in diesem Beispiel vorhanden, die Sie bisher noch nicht gesehen haben. Nachdem die Namespaces mit *open*-Anweisungen verfügbar gemacht worden sind, wird eine Funktion namens *loadWindow* deklariert. Die Methode *Application.LoadComponent* lädt den Inhalt der XAML-Datei in Listing 1.3 und erzeugt daraus ein Objekt vom Typ *Window*. Innerhalb dieses Objekts wird nach dem Button gesucht, der im XAML definiert wurde, um einen Event Handler für dessen Event *Add* anzuhängen. Dieser Event Handler ist in Form eines Lambda-

Ausdrucks implementiert, der die Nachricht *Hallo Welt* mithilfe einer *MessageBox* auf dem Bildschirm anzeigt. Die letzte Zeile auch in diesem Programm ist ein Aufruf an die Infrastruktur der Plattform, um das Programm zu starten.

In der vorletzten Zeile kann der aufmerksame Leser noch ein interessantes Detail erkennen: Hier wird das Attribut *STAThread* angewandt, das Sie eventuell aus anderen .NET-Sprachen kennen. Wenn Sie mit dem Attribut vertraut sind, wissen Sie sicherlich, das dieses auf der Hauptmethode eines ausführbaren Programms angewandt werden muss – dies lässt vermuten, dass die letzte Zeile des obigen Programms von F# als Hauptmethode angesehen wird. Dies ist eine korrekte Annahme: Der alleinstehende Code in diesem Programm, der nicht mit der Deklaration der Funktion *loadWindow* zu tun hat, wird als automatisch startbares Hauptprogramm interpretiert.

1.1.4 Die Entwicklungsumgebung(en)

F# ist im Hause Microsoft möglicherweise die unabhängigste Programmiersprache. Das Team, das an der Sprache arbeitet, ist nicht nur an der Entwicklung der Sprache für Visual Studio beteiligt, sondern unterstützt auch aktiv die Verwendung von F# außerhalb von Microsofts IDE. Damit ist F# sehr vielfältig verwendbar: als Sprache innerhalb von Visual Studio, seinen Projekten und Solutions, aber auch in jeder anderen Umgebung, die .NET unterstützt.

Wenn Sie mit Visual Studio arbeiten, bekommen Sie seit Version 2010 einige Vorlagen mitgeliefert, mit deren Hilfe sie verschiedene Projekte in der Sprache F# automatisch erzeugen können. In dem Dialog, den Visual Studio bei der Erzeugung eines neuen Projekts angezeigt, steht die Sprache gleichberechtigt neben althergebrachten .NET-Sprachen wie C#, Visual Basic oder C++. Leider ist die Sammlung der verfügbaren Vorlagen noch nicht so vollständig wie bei diesen anderen Sprachen, sodass sie für komplexere Projekte oft etwas manuelle Arbeit leisten müssen.