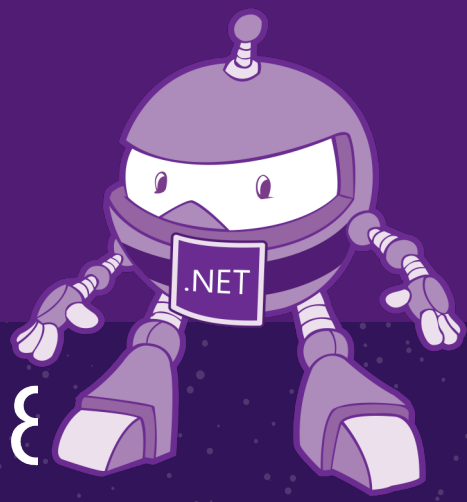

.NET Conf 2018
Discover the world of .NET
dotnet new
.NET

What's new in F# 4.5

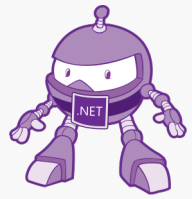
Phillip Carter



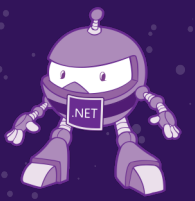
.NET Conf 2018

Overview

- New F# 4.5 features
- How to get started



New F# 4.5 features

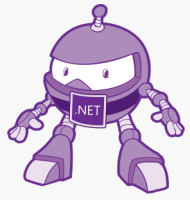


Quick list

- Versioning alignment
- Span<‘T> and friends
- New keyword: match!
- Relaxed indentation and upcast requirements
- Better async stack traces
- ValueOption<‘T>
- Small goodies in FSharp.Core

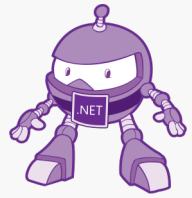
Versioning alignment

- F# 4.1
  - F# language version **4.1**
  - FSharp.Core NuGet package version **4.3.4**
  - FSharp.Core binary version **4.4.1.0**
- F# 4.5
  - F# language version **4.5**
  - FSharp.Core NuGet package version **4.5.x**
  - FSharp.Core binary version **4.5.0.0**
- Finally, they have the same first two numbers



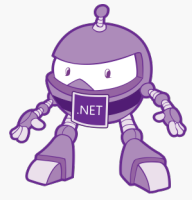
Span<‘T> and friends

- It’s actually 8 new features
  - inref<‘T> and outref<‘T> types
  - byrefs as a concept fleshed out
  - Production and consumption of byref returns
  - Extension method support for byrefs
  - Comprehensive safety checks for byrefs and byref-like types
  - IsByRefLike structs
  - IsReadOnly structs
  - voidptr type and NativePtr.ofVoitPtr/NativePtr.toVoidPtr functions
- Span<‘T> is a byref-like type



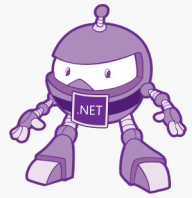
Byrefs

- inref<‘T>/byref<‘T>/outref<‘T>
  - Managed pointer types
  - inref<‘T> is readonly
  - byref<‘T> is read/write
  - outref<‘T> signals write-only
- Strong safety guarantees
  - Cannot be used outside the scope they are defined in
  - Cannot be contained within a heap-allocated type
  - Cannot be captured by any closure construct
  - Cannot be used as a generic type parameter



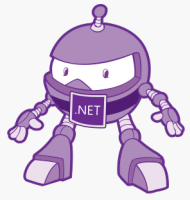
Byref returns

- F# 4.1 - only consumption supported
- F# 4.5 – production and consumption supported
  - Return value is implicitly dereferenced when consumed
  - The ‘x’ in ‘for x in …’ loops is also implicitly dereferenced
    - Example: looping through a Span<‘T>
  - Passing a returned reference by reference is done with ‘&’



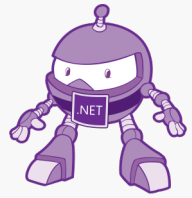
Byref-like Structs

- Use [<IsByRefLike>] attribute
  - Can be applied to F# structs
  - Can be applied to struct Records and Discriminated Unions
- Stack-only value type with byref semantics
  - Cannot be contained within a heap-allocated type
    - Class, normal struct, non-struct Record, or non-struct Discriminated Union
  - Cannot be used as the type of a Record or DU case
  - Cannot be captured by a closure construct
  - Cannot be used as a generic type parameter
- Span<‘T> is a byref-like struct



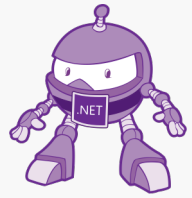
Readonly structs

- Use [<IsReadOnly>] attribute
  - Can be applied to F# structs
  - Can be applied to struct Records and Discriminated Unions
- Informs C# callers the struct is readonly
  - Public members and ‘this’ pointer cannot be modified by C#
  - F# structs have been readonly for F# consumers prior to this
- ReadOnlySpan<‘T> is a readonly byref-like struct



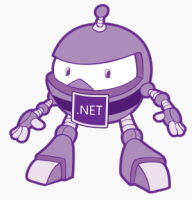
Void pointers

- New type: voidptr
  - Untyped, unmanaged pointer in F# code
  - Useful for interop with native code
- NativePtr functions
  - NativePtr.ofVoidPtr -> convert voidptr to nativeptr<‘T>
  - NativePtr.toVoidPtr -> convery nativeptr<‘T> to voidptr



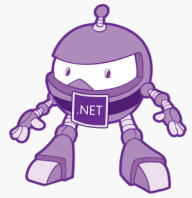
match! keyword

- New keyword for computation expressions
  - Community contribution (John Wostenberg)
  - Simplify pattern matching inside a computation expression
  - Often used with async code that returns an F# option



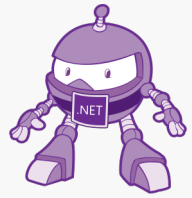
Relaxations in your code

- upcast with yield in F# seq/list/array expressions
  - Use of ‘:>’ no longer needed to use a supertype with yield
- Indentation for list and array expressions
  - Particularly with named arguments in method calls
  - Inspired by Fable/Elmish-style UI programming



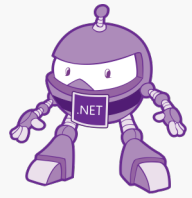
ValueOption<‘T>

- Like Option<‘T>, but a struct
  - voption keyword for type signatures
  - Building block for future work to allow active patterns to be struct-based



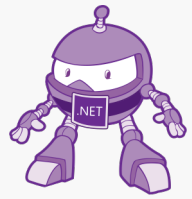
Better async stack traces

- Before
  - Information in stack traces was unusable
    - No user code
    - No user line numbers
- After
  - User code names
  - User line numbers

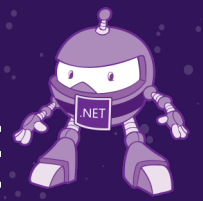


Additional FSharp.Core items

- FuncConvert APIs
  - FromAction and ToAction with various overloads
  - Helps interop between C# Actions to F# functions
- Map.TryGetValue
  - Similar to TryParse APIs in F#
    - Takes in byref and returns bool, is converted to tuple for F# callers
- General improvements and bug fixes
  - Thanks to the F# community to their work here!

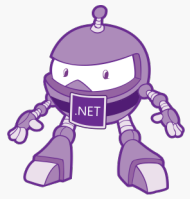


How to get started

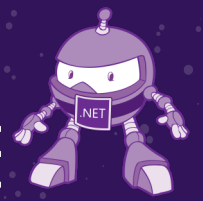


Use F# 4.5 today

- aka.ms/fsharphome
- Install .NET Core (latest)
- Use Visual Studio 2017 Update 15.8
  - Comes with F# 4.5 by default
  - Also installs .NET Core if you don't have it already
- Visual Studio Code with Ionide plugin
  - Uses your .NET Core installation
- Visual Studio for Mac
  - Uses your .NET Core installation



F# vNext



.NET Conf 2018

Next time, on F#

- Nullability for reference types
  - Reference types default to non-null
  - Backwards compatible and tunable
- Anonymous Record Types
  - Ad-hoc, named groupings of data
  - Do away with boilerplate
- task { } workflow
- Many more considerations on GitHub
  - github.com/fsharp/fslang-suggestions