

Expert F# Errata, 26 Jan 2009

Contents

1. When using the F# September 2008 CTP.....	2
2. Corrections and Errors	3
Foreword, p. xxii.....	3
Chapter 2, p. 7.....	3
Chapter 2, p. 16.....	3
Chapter 3, p. 29.....	3
Chapter 3, p. 29.....	4
Chapter 3, p. 31., table 3-5	4
Chapter 3, p. 47.....	4
Chapter 3, p. 55.....	4
Chapter 3, p. 55.....	4
Chapter 4, p. 73.....	4
Chapter 4, p. 74.....	5
Chapter 4, p. 82.....	5
Chapter 4, p. 84.....	5
Chapter 4, p. 85.....	5
Chapter 5, p. 107. – 05/09/2008.....	5
Chapter 5, p. 107. – 05/09/2008.....	5
Chapter 6, p. 136, Listing 6-3	5
Chapter 6, p. 136.....	6
Chapter 6, p. 130.....	6
Chapter 6, p. 141.....	6
Chapter 6, p. 142.....	6
Chapter 7, p. 168.....	6
Chapter 7, p. 171, and Table of Contents	6
Chapter 8, p. 182, Listing 8-1	6
Chapter 8, p. 182, Listing 8-2	6
Chapter 8, p. 189.....	7
Chapter 8, p. 202, Listing 8-9	7
Chapter 9, p. 225.....	7

Chapter 9, p. 238.....	7
Chapter 9, p. 251.....	8
Chapter 10, p. 256.....	10
Chapter 10, p. 256.....	10
Chapter 10, p. 268.....	10
Chapter 11, p. 295, Listing 11-2	10
Chapter 11, p. 305.....	10
Chapter 12, p. 327.....	10
Chapter 13, p. 374, Table 13-3.....	10
Chapter 13, p. 384.....	10
Chapter 14, p. 423.....	11
Chapter 15, p. 438.....	11

1. When using the F# September 2008 CTP

The function `String.split` is now defined in the F# Power Pack. You must add

```
#r "FSharp.PowerPack.dll";;
#r "FSharp.PowerPack.Compatibility.dll";;
```

The following minor F# design changes or renamings are relevant and should be applied to the code in the book:

```
(type ty)      →      typeof<ty>
fold1_left    →      reduce_left
string.Empty  →      System.String.Empty
Seq.generate   →      SequenceExpressionHelpers.generate
Seq.generate_using
               →      SequenceExpressionHelpers.generate_using

open Microsoft.FSharp.Math.Primitives
               →      open Microsoft.FSharp.Math

open Microsoft.FSharp.Control.Mailboxes
               →      open Microsoft.FSharp.Control
```

```
e |> Seq.untyped_iter f
      → e |> Seq.cast |> Seq.iter f

x.Iterate(f) → x |> List.iter f
```

In Chapter 10, the line

```
"All of these DLLs are automatically referenced...in F# projects"
```

is not strictly correct when using the F# September 2008 CTP and later versions. Instead, these are referenced when using the "fsc.exe" command line program, but not when working from within a Visual Studio project.

2. Corrections and Errors

Foreword, p. xxii.

A name is misspelt. Luckily they got it right on the cover.

Chapter 2, p. 7.

The function `String.split` is now defined in the F# Power Pack. You must add

```
#r "FSharp.PowerPack.dll";;

#r "FSharp.PowerPack.Compatibility.dll";;
```

Chapter 2, p. 16.

The code fragment shows this output:

```
> Set.of_list ["b";"a";"b";"b";"c" ];;
val it : Set<string> = set [ "a"; "b"; "c" ]
```

Actually F# interactive shows this:

```
val it : Set<string> = seq ["a"; "b"; "c"]
```

i.e. "seq" instead of "set"

Chapter 3, p. 29.

In the eBook, the list of operators shows "o umlaut" three times instead of ^{^^^} on at least some computers

Chapter 3, p. 29.

The function `doubleAndAdd` should be called `squareAndAdd`

Chapter 3, p. 31., table 3-5

The sample use of hyperbolic trigonometric functions should be "cosh 1.0" rather than "cos 1.0"

Chapter 3, p. 47.

When using the F# September 2008 CTP, the definition of "truncate" has changed to have type

```
truncate : float -> float
```

As a result the code in this sample needs adjusting, e.g. to

```
let remap (r1 : Rectangle) (r2 : Rectangle) =  
    let scalex = float r2.Width / float r1.Width  
    let scaley = float r2.Height / float r1.Height  
    let mapx x = int (float r2.Left + truncate( float (x - r1.Left) * scalex))  
    let mapy y = int (float r2.Top + truncate( float (y - r1.Top) * scaley))  
    let mapp (p:Point) = Point(mapx p.X, mapy p.Y)  
    mapp
```

Chapter 3, p. 55.

The result of `seq {1I .. 1000000000I}` is stated as :

```
val it : seq<bigint> = seq [ 0I; 1I; 2I; 3I; ...]
```

It should be stated as

```
val it : seq<bigint> = seq [ 1I; 2I; 3I; 4I; ...]
```

Chapter 3, p. 55.

A construct used on this page looks likely to be deprecated in a future release of F#. The folks over at Microsoft Research tell us a replacement construct called 'linspace' will be included instead. The current F# compiler gives:

```
seq {-100.0 .. 100.0};;  
-----^
```

`stdIn(3,4): warning: Floating point ranges are deprecated because inherent imprecisions in floating point computations give unintuitive results. Use an integer range instead and to floating point using 'float' or 'float32'.`

Chapter 4, p. 73

"read the cell, and mutate the cell, respectively."

should read:

“mutate the cell, and read the cell, respectively.”

Chapter 4, p. 74

Typo in the title “Which Data Structures Are Mutable?”

Chapter 4, p. 82

```
val it : bool : true
```

should be:

```
val it : bool = true
```

and

```
val it : bool : true
```

should be:

```
val it : bool = true
```

Chapter 4, p. 84.

```
val sparseMap : Dictionary <string, string list>
```

should be:

```
val sparseMap : Dictionary <(int * int), float>
```

Chapter 4, p. 85.

The sample

```
raise (System.InvalidOperationException("not today thank you"));
```

does not typecheck when entered into F# Interactive. It hits the value restriction (see Chapter 5) because the overall type is not known. Simply constrain the type:

```
(raise (System.InvalidOperationException("not today thank you"))) : int;;
```

Chapter 5, p. 107. – 05/09/2008

The text:

```
val writeValue : #System.IO.Stream -> 'a -> obj
```

should be:

```
val writeValue : #System.IO.Stream -> 'a -> ()
```

Chapter 5, p. 107. – 05/09/2008

The text:

```
unbox x
```

should be:

```
unbox res
```

Chapter 6, p. 136, Listing 6-3

The title of this list is not correct

Chapter 6, p. 136

```
[<OverloadID("subtractPointVector")>]...
```

The non-capitalized overload id here is not intentional, though does not matter.

Chapter 6, p. 130

The text:

```
Rectangle(center.X - side, center.Y - side, side*2, side*2) }  
would be more accurate as
```

```
Rectangle(center.X - side/2, center.Y - side/2, side, side) }  
though it is of no real substance to the discussion
```

Chapter 6, p. 141.

The text:

```
> (circle2 :> IShape).BoundingBox;;  
val it : Rectangle = {X=0,Y=0,Width=3,Height=3}
```

should be:

```
> (circle2 :> IShape).BoundingBox;;  
val it : Rectangle = {X=-10,Y=-10,Width=21,Height=21}
```

Chapter 6, p. 142.

The text:

```
> smallSquare.BoundingBox;;  
val it : Rectangle = {X=0,Y=0,Width=3,Height=3}
```

should be:

```
> smallSquare.BoundingBox;;  
val it : Rectangle = {X=0,Y=0,Width=2,Height=2}
```

Chapter 7, p. 168

The text:

```
let spotted = Fictional.whales.[Int32.of_string idx]
```

should be

```
let spotted = Fictional.whales.[idx]
```

Chapter 7, p. 171, and Table of Contents

"and the Using" should be "and Using"

Chapter 8, p. 182, Listing 8-1

The code shown uses a property `IsZero` – this should be called `isZero`.

Chapter 8, p. 182, Listing 8-2

Since publication, the technique of using `override` and `interface` implementations in augmentations is no longer “good” F# practice and we have been notified that later versions of the F# compiler may give

warnings or errors when this technique is used. Hence we recommend you use the technique shown in Listing 8-1.

Chapter 8, p. 189

In the note, there is a reference to the discussion of thread synchronization in chapter 14. The discussion is actually in chapter 13.

Chapter 8, p. 202, Listing 8-9

"loop chain" should be "loop chain 0" – we were not passing the initial accumulator to the function

Chapter 9, p. 225

In the description of complex numbers, magnitude and phase are the wrong way around.

Chapter 9, p. 238

In the in the definition of MBuilder,

```
member b.Bind(v,f) = bindM p f
```

should be member

```
member b.Bind(v,f) = bindM v f
```

Chapter 9, ~p. 245

The SchemaReader code uses the "old" F# Reflection API. The updated code for the finalized API is below

```
open System
open System.IO
open Microsoft.FSharp.Reflection

/// An attribute to be added to fields of a schema record type to indicate the
/// column used in the data format for the schema.
type ColumnAttribute(col:int) =
    inherit Attribute()
    member x.Column = col

/// SchemaReader builds an object that automatically transforms lines of text
/// files in comma-separated form into instances of the given type 'Schema.
/// 'Schema must be an F# record type where each field is attributed with a
/// ColumnAttribute attribute, indicating which column of the data the record
/// field is drawn from. This simple version of the reader understands
/// integer, string and DateTime values in the CSV format.
type SchemaReader<'Schema>() =

    // Grab the object for the type that describes the schema
    let schemaType = typeof<'Schema>

    // Grab the fields from that type
    let fields = FSharpType.GetRecordFields(schemaType)

    // For each field find the ColumnAttribute and compute a function
    // to build a value for the field
    let schema =
        fields |> Array.map (fun fldIdx fld ->
            let fldInfo = schemaType.GetProperty(fld.Name)
            let fldConverter =
                match fld.PropertyType with
```

```

| ty when ty = typeof<string> -> (fun (s:string) -> box s)
| ty when ty = typeof<int> -> (System.Int32.Parse >> box)
| ty when ty = typeof<DateTime> -> (System.DateTime.Parse >> box)
| ty -> failwithf "Unknown primitive type %A" ty

let attrib =
    match fieldInfo.GetCustomAttributes(typeof<ColumnAttribute>,
                                        false) with
    | [] (:? ColumnAttribute as attrib) [] -> attrib
    | _ -> failwithf "No column attribute found on field %s" fld.Name
        (fldIdx, fld.Name, attrib.Column, fieldConverter))

// Compute the permutation defined by the ColumnAttributes indexes
let columnToFldIdxPermutation c =
    schema |> Array.pick (fun (fldIdx, _, colIdx, _) ->
        if colIdx = c then Some fldIdx else None)

// Drop the parts of the schema we don't need
let schema =
    schema |> Array.map (fun (_, fldName, _, fldConv) -> (fldName, fldConv))

// Compute a function to build instances of the schema type. This uses an
// F# library function.
let objectBuilder = FSharpValue.PreComputeRecordConstructor(schemaType)

// OK, now we're ready to implement a line reader
member reader.ReadLine(textReader: TextReader) =
    let line = textReader.ReadLine()
    let words = line.Split([|', '|]) |> Array.map(fun s -> s.Trim())
    if words.Length <> schema.Length then
        failwithf "unexpected number of columns in line %s" line
    let words = words |> Array.permute columnToFldIdxPermutation

    let convertColumn colText (fieldName, fieldConverter) =
        try fieldConverter colText
        with e ->
            failwithf "error converting '%s' to field '%s'" colText fieldName

    let obj = objectBuilder (Array.map2 convertColumn words schema)

// OK, now we know we've dynamically built an object of the right type
unbox<'Schema>(obj)

// OK, this read an entire file
member reader.ReadFile(file) =
    seq { use textReader = File.OpenText(file)
        while not textReader.EndOfStream do
            yield reader.ReadLine(textReader) }

```

Chapter 9, p. 251

The Quotation error estimation code uses the "old" quotation API. The updated code for the finalized API is below

open Microsoft.FSharp.Quotations

```
open Microsoft.FSharp.Quotations.Patterns
open Microsoft.FSharp.Quotations.DerivedPatterns
```

```
type Error = Err of float
```

```
let rec errorEstimateAux (e:Expr) (env : Map<Var, _>) =
  match e with
  | SpecificCall <@@ (+) @@> (tyargs, _, [xt; yt]) ->
    let x, Err(xerr) = errorEstimateAux xt env
    let y, Err(yerr) = errorEstimateAux yt env
    (x+y, Err(xerr+yerr))

  | SpecificCall <@@ (-) @@> (tyargs, _, [xt; yt]) ->
    let x, Err(xerr) = errorEstimateAux xt env
    let y, Err(yerr) = errorEstimateAux yt env
    (x-y, Err(xerr+yerr))

  | SpecificCall <@@ ( * ) @@> (tyargs, _, [xt; yt]) ->
    let x, Err(xerr) = errorEstimateAux xt env
    let y, Err(yerr) = errorEstimateAux yt env
    (x*y, Err(xerr*abs(y)+yerr*abs(x)+xerr*yerr))

  | SpecificCall <@@ abs @@> (tyargs, _, [xt]) ->
    let x, Err(xerr) = errorEstimateAux xt env
    (abs(x), Err(xerr))

  | Let(var, vet, bodyt) ->
    let varv, verr = errorEstimateAux vet env
    errorEstimateAux bodyt (env.Add(var, (varv, verr)))

  | Call (None, MethodWithReflectedDefinition(Lambda(v, body)), [arg]) ->
    errorEstimateAux (Expr.Let(v, arg, body)) env

  | Var(x) -> env.[x]

  | Double(n) -> (n, Err(0.0))

  | _ -> failwithf "unrecognized term: %A" t
```

```
let rec errorEstimateRaw (t : Expr) =
  match t with
  | Lambda(x, t) ->
    (fun xv -> errorEstimateAux t (Map.ofSeq [(x, xv)]))
  | PropertyGet (None, PropertyGetterWithReflectedDefinition(body), []) ->
    errorEstimateRaw body
  | _ -> failwithf "unrecognized term: %A - expected a lambda" t
```

```
let errorEstimate (t : Expr<float -> float>) = errorEstimateRaw t
```

The error estimation for multiplication and division is not calculated correctly. For multiplication, the error should be

$$\text{abs}(x) * \text{yerr} + \text{abs}(y) * \text{xerr} + \text{xerr} * \text{yerr}$$

For division, the error should be

$$(xa*yerr+ya*xerr)/(ya*(ya-yerr))$$
 where $xa=abs(x)$ and $ya=abs(y)$

Chapter 10, p. 256

The example of an embedded reference in source code is missing a closing quote

```
#! @"C:\Program Files\Reference Assemblies\Microsoft\Framework\v3.5";
```

Chapter 10, p. 256

The line

"All of these DLLs are automatically referenced...in F# projects"

is not strictly correct when using the F# September 2008 CTP and later versions. Instead, these are referenced when using the "fsc.exe" command line program, but not when working from within a Visual Studio project.

Chapter 10, p. 268.

The result output for the matrix sample got mangled: we changed the inputs but didn't replace the printed output! Enter the sample into F# Interactive and you'll get the right answer ☺ Think of it as a test for your school arithmetic.

Chapter 11, p. 295, Listing 11-2

There is a naming inconsistency here – the variable name should be "maxVisibleValue"

```
let mutable minVisibleValue = Single.Max (and the same in the next 3 lines)
```

Chapter 11, p. 305.

Jeffrey Sax says: *This is a technicality: A point is in the Mandelbrot set if the sequence z_i does not diverge to infinity. Most sequences for points in the Mandelbrot set don't converge to a point but to a cycle of points. In fact, the "little" Mandelbrot sets you see everywhere are attractors for cycles of different lengths.*

Chapter 12, p. 327

`Simplify in ExprUtil.fs` does not check whether the divisor is an expression that simplifies to zero.

Chapter 13, p. 374, Table 13-3

The signature of `Async.Parallel` is actually

```
#seq<Async<'a>> -> Async<'a[]>
```

not

```
Async<#seq<'a>> -> Async<'a[]>
```

Chapter 13, p. 384

When using the F# 2008 September CTP,

open Microsoft.FSharp.Control.Mailboxes
should be

open Microsoft.FSharp.Control

Chapter 14, p. 423.

The link to F# Web Tools should read <http://www.codeplex.com/fswebtools>.

Chapter 15, p. 438.

The table numbers in this chapter are wrong. Also, the table titled "Common Databases" does not mention DB2, which is available at

<http://www-306.ibm.com/software/data/db2/windows/dotnet.html>.