Semantic Versioning for .NET libraries and NuGet packages (C#/F#)

MF#K November 2016 Meetup

@Prosa 2016-11-29

Overview

- About me
- Semantic Versioning
- elm-package ***bump*** and ***diff***
- ***SpiseMisu.SemanticVersioning*** library
- Demo
- Q & A

About me (very shortly)

- Ramón Soto Mathiesen
- MSc. Computer Science ***DIKU/Pisa*** and minors in Mathematics ***HCØ***
- ***CompSci*** @ SPISE MISU ApS
  - ***"If I have seen further it is by standing on the shoulders of giants"***
    -- Isaac Newton (Yeah Science, Bitch … Mostly mathematics)
  - ***Elm*** with a bit of ***Haskell*** and a bit of ***F#*** (fast prototyping)
- Elm / Haskell / TypeScript / F# / OCaml / Lisp / C++ / C# / JavaScript
- Blog: http://blog.stermon.com/

Semantic Versioning (SemVer)

- *“In the world of software management there exists a dread place called **dependency hell**”*
  - *“The bigger your system grows and the more packages you integrate into your software, the more likely you are to find yourself in it”*
- If dependencies are specified too loosely, you will probably end up breaking your build more than desired
- So how to solve this? With a few rules, ***enforced by*** ~~documentation or~~ ***the code itself***, …

Semantic Versioning (SemVer)

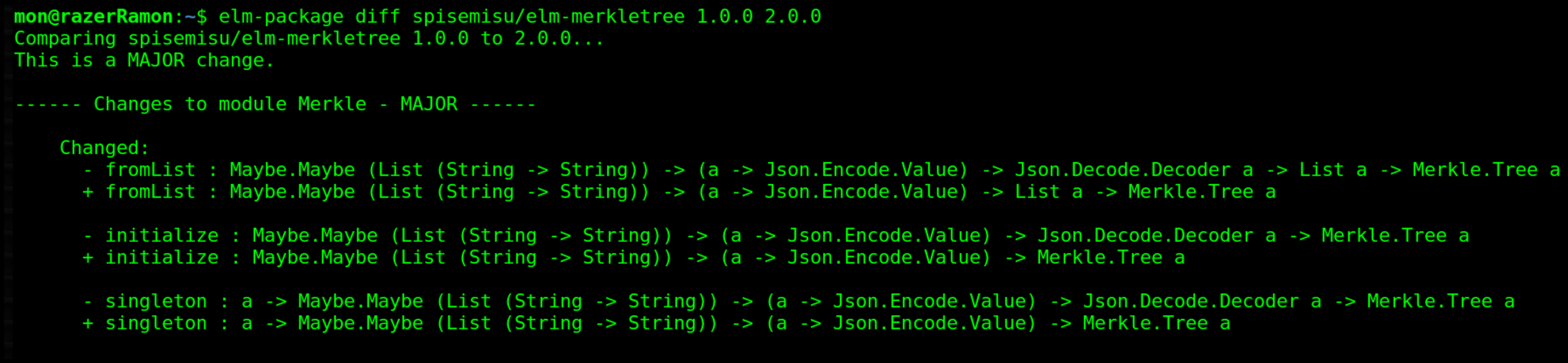
- … given a version number (***MAJOR.MINOR.PATCH***), increment the:
  - ***MAJOR*** version when you make incompatible API changes,
  - ***MINOR*** version when you add functionality in a backwards-compatible manner, and
  - ***PATCH*** version when you make backwards-compatible bug fixes
- Source: http://semver.org/

elm-package bump

- Elm package version rules:
  - Versions all have exactly three parts: MAJOR.MINOR.PATCH ✓
  - All packages start with initial version 1.0.0 ✓
  - Versions are incremented based on how the API changes:
    - PATCH - the API is the same, no risk of breaking code ✓
    - MINOR - values have been added, existing values are unchanged ✓
    - MAJOR - existing values have been changed or removed ✓
- ***elm-package*** will ***bump*** versions for you, ***automatically enforcing*** these ***rules*** ✓ ✓ ✓ (DING DING DING MF#K)

elm-package diff (+bump)

- Show the changes between versions:



```
mon@razerRamon:~$ elm-package diff spisemisu/elm-merkletree 1.0.0 2.0.0
Comparing spisemisu/elm-merkletree 1.0.0 to 2.0.0...
This is a MAJOR change.

------ Changes to module Merkle - MAJOR ------

    Changed:
      - fromList : Maybe.Maybe (List (String -> String)) -> (a -> Json.Encode.Value) -> Json.Decode.Decoder a -> List a -> Merkle.Tree a
      + fromList : Maybe.Maybe (List (String -> String)) -> (a -> Json.Encode.Value) -> List a -> Merkle.Tree a

      - initialize : Maybe.Maybe (List (String -> String)) -> (a -> Json.Encode.Value) -> Json.Decode.Decoder a -> Merkle.Tree a
      + initialize : Maybe.Maybe (List (String -> String)) -> (a -> Json.Encode.Value) -> Merkle.Tree a

      - singleton : a -> Maybe.Maybe (List (String -> String)) -> (a -> Json.Encode.Value) -> Json.Decode.Decoder a -> Merkle.Tree a
      + singleton : a -> Maybe.Maybe (List (String -> String)) -> (a -> Json.Encode.Value) -> Merkle.Tree a
```

Rust and others should as well

- Rust (suggestion for cargo):
  - Signature based API comparison
- Haskell (why does cabal or stack not have this?):
  - semver-0.3.3.1

Note: We all tried to use a given package that failed to install due to issues with dependent packages right? Frustration most of the time tend to dropping a given package and sometimes even moving on to other languages ...

SpiseMisu.SemanticVersioning

- My proposal of SemVer for .NET libraries as well as for NuGet packages
  - Support for both C#/F# (VB? Say JUAT?)
- As with Elm, I would like the ***rules to be enforcement by the code itself***, instead of by humans. Otherwise we would be back to square one as humans tend to fail with repetitive task
- ***Elm*** has it ***easy*** as ***everything is Open Source***, therefore source code can be parsed while with .NET (proprietary libraries) ...

SpiseMisu.SemanticVersioning

- Handle cases like the Fsharp.Core does (Reflection):
  - FSharp.Core.Unittests
    - LibraryTestFx.fs#L93
    - LibraryTestFx.fs#L103-L110

```
asm.GetExportedTypes()
...

(* extract canonical string form for every public member of every type *)
seq {
    yield! t.GetRuntimeEvents()     |> Seq.filter (fun m -> m.AddMethod.IsPublic) |> Seq.map cast
    yield! t.GetRuntimeProperties() |> Seq.filter (fun m -> m.GetMethod.IsPublic) |> Seq.map cast
    yield! t.GetRuntimeMethods()    |> Seq.filter (fun m -> m.IsPublic) |> Seq.map cast
    yield! t.GetRuntimeFields()     |> Seq.filter (fun m -> m.IsPublic) |> Seq.map cast
    yield! ti.DeclaredConstructors  |> Seq.filter (fun m -> m.IsPublic) |> Seq.map cast
    yield! ti.DeclaredNestedTypes   |> Seq.filter (fun ty -> ty.IsNestedPublic) |> Seq.map cast
} |> Array.ofSeq
```

SpiseMisu.SemanticVersioning

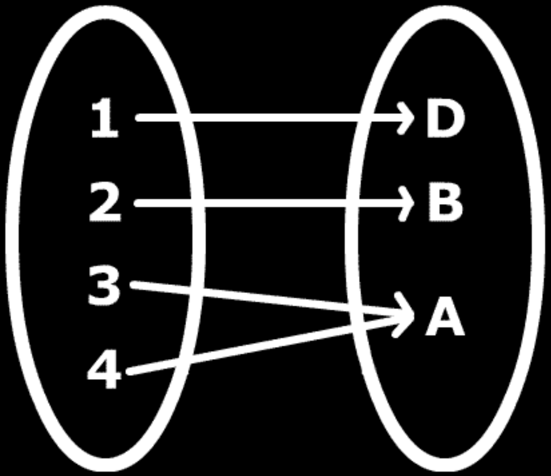
- Handle cases like the Fsharp.Core does ... (+ more):
  - The main issue with ***basic Reflection***, is that it ***works great with C#*** libraries, but ***not so much with F#***. The following functions signatures are represented on the same way in .NET canonical form (no curried arguments):

```
let foo (x,y) = x + y
let bar x y = x + y

(* both represented as *)
x:System.Int32 * y:System.Int32 -> z:System.Int32

(* but should be respectively *)
x:System.Int32 * y:System.Int32 -> z:System.Int32
x:System.Int32 -> y:System.Int32 -> z:System.Int32
```

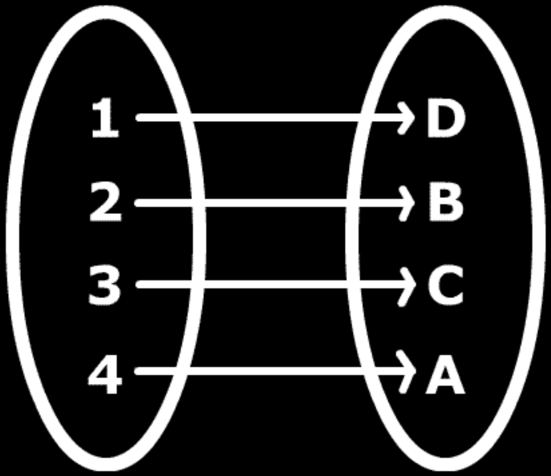
SpiseMisu.SemanticVersioning

- Handle cases like the Fsharp.Core does … (+ more):
  - Other constructs such as ***Product Types***, ***Modules*** and even ***Enums*** & ***Sum Types*** (due to ***pattern matching***) needs to be handled in a special way:
    - Cases like ***Active/Partial Patterns*** and ***MeasureOfUnits*** are not handled (yet? Is it even necessary?)
    - Please look into the Open Source code to see what is done for each case
- ***Main goal*** is to ***create a bijective function*** that would ***replace*** the ***non-injective and surjective function*** which will ***ensure*** that a given ***input value*** will always have a ***unique output value***. Think of it as a perfect hash function with no collisions

SpiseMisu.SemanticVersioning



non-injective and surjective



bijective

SpiseMisu.SemanticVersioning

- Similar readability as Haskell and Elm signatures (last type is the return value while the others are input parameters). Example:

```
FooBar : Foo → (Bar * Baz) → Qux
```

Note: This is also why I now write F# code like this:

```
let foobar : int → (int * int)→ int =
  fun x (y,z) → x + y + z
```

SpiseMisu.SemanticVersioning

- .NET Library (Assembly):
  - Is usually a single file compiled to target a specific version of the .NET Framework. Example:

    mscorlib,Version=4.0.0.0, Culture=neutral,PublicKeyToken=…
- .NET NuGet package:
  - Is a unit of distribution containing some metadata as well as binaries. In ***most*** cases, there are binaries targeting several versions of the .NET Framework.

    Note: We are only interested in libraries (***lib/…/*.ddl***)

SpiseMisu.SemanticVersioning .NET NuGet package

```fsharp
#!/usr/bin/env fsharpi

#I @"./SpiseMisu.SemanticVersioning/"
#r @"SpiseMisu.SemanticVersioning.dll"

open System
open System.Diagnostics
open System.Reflection

open SpiseMisu

let pkgid = @"Newtonsoft.Json"

let assembly =
  Assembly.LoadFile
    @"./packages/Newtonsoft.Json.7.0.1/lib/net45/Newtonsoft.Json.dll"

Semantic.Versioning.nugetbump
  pkgid
  NuGet.dotNet.Net45
  assembly
|> printfn "%s"

Semantic.Versioning.nugetdiff
  pkgid
  NuGet.dotNet.Net45
  (Some "7.0.1")
  pkgid
  NuGet.dotNet.Net45
  None
|> Array.iter(printfn "%s")
```

SpiseMisu.SemanticVersioning .NET Library (Assembly)

```
#!/usr/bin/env fsharpi

#I @"./SpiseMisu.SemanticVersioning/"
#r @"SpiseMisu.SemanticVersioning.dll"

open System
open System.Diagnostics
open System.Reflection

open SpiseMisu

let released =
  Assembly.LoadFile
    @"./packages/Newtonsoft.Json/lib/net45/Newtonsoft.Json.dll"
let modified =
  Assembly.LoadFile
    @"./packages/Newtonsoft.Json.7.0.1/lib/net45/Newtonsoft.Json.dll"

Semantic.Versioning.asmbump released modified
|> printfn "%s"

Semantic.Versioning.asmdiff released modified
|> Array.iter(printfn "%s")
```

SpiseMisu.SemanticVersioning .NET Library (documentation)

```
#!/usr/bin/env fsharpi

#I @"./SpiseMisu.SemanticVersioning/"
#r @"SpiseMisu.SemanticVersioning.dll"

open System
open System.Diagnostics
open System.Reflection

open SpiseMisu

let assembly =
  Assembly.LoadFile
    @"./packages/Newtonsoft.Json/lib/net45/Newtonsoft.Json.dll"

Semantic.Versioning.markdown assembly
|> Array.iter(printfn "%s")
```

SpiseMisu.SemanticVersioning .NET Library (raw)

```fsharp
#!/usr/bin/env fsharpi

#I @"./SpiseMisu.SemanticVersioning/"
#r @"SpiseMisu.SemanticVersioning.dll"

open System
open System.Diagnostics
open System.Reflection

open SpiseMisu

let assembly =
  Assembly.LoadFile
    @"./packages/Newtonsoft.Json/lib/net45/Newtonsoft.Json.dll"

Semantic.Versioning.raw assembly
|> Set.toArray
|> Array.iter(fun (prefix, body) -> printfn "%s - %s" prefix body)
```

Demo

What’s next?

- Publish a blog post for F# Advent Calendar 2016
- Release Open Source library @ GitHub
- Review of code by .NET experts

What’s next?

- Integrated in:
  - NuGet (or something similar, please ***steal with pride***)
  - FAKE
  - Paket
- To catch on with C# Community, it has to be ***totally transparent*** and with ***no F# related stuff***. Therefore I will need a standalone executable (something like ***paket.exe***)

  Note: I’m thinking about using Mono mkbundle

Summary

- Semantic Versioning
  - Set of Rules
- elm-package ***bump*** and ***diff***
  - SemVer rules enforced by the code itself
- ***SpiseMisu.SemanticVersioning*** library
  - Support for Assemblies and NuGet as well as C#/F# (even proprietary due to Reflection)
  - SemVer rules are also enforced by the code itself, just like elm-package
  - Output is markdown
- Demo
- What's next?
  - Open Source library as well as standalone binary
  - Integration with NuGet, FAKE, Paket

Q & A

Any Questions?