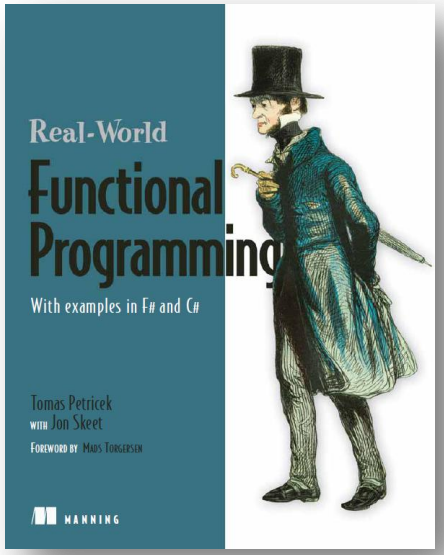
COPENHAGEN
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE 2011
Training: May 9-10 // Conference : May 11-13
goto; conference

Accessing loosely structured data from F# and C#

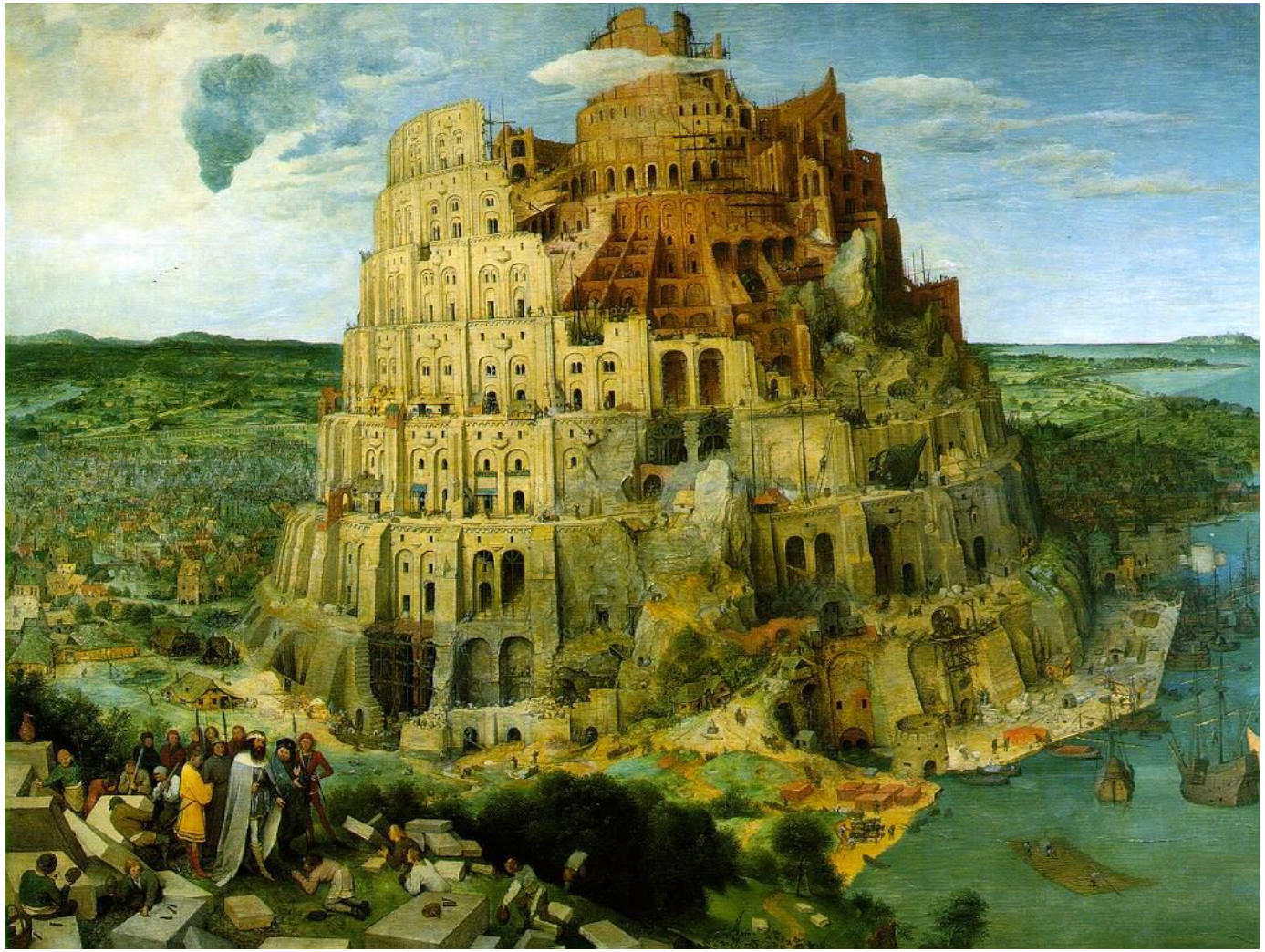
Tomas Petricek
PhD Student
Microsoft C# MVP
http://tomasp.net/blog | @tomaspetricek

A little bit about me…

- Real World Functional Programming
  - Co-authored by Jon Skeet
  - For C# developers
- Worked on F# at MSR
  - Internships & Contracting
  - Blogged about F# in 2006
- PhD Student at University of Cambridge



The Problem



The Problem

- Structure in the language
  - Classes with properties (C#)
  - Data types or classes (F#)
- Structure in the data source
  - Database structure, XML schema (explicit)
  - REST service or JSON file (implicit)
- How to solve the impedance mismatch?

What you'll learn today

- "Design Patterns" for solving structure mismatch
  - What options do we have in general
- Interesting dynamic and reflection tricks
  - Things you can implement in C# or F#
- How F# type providers work
  - Future F# technology that "brings data into the language, strongly typed"

- Expression scale
  - Using dynamic type or operator
- Program scale
  - Defining structure in the language
- Internet scale
  - Generate structure using type providers

Dynamic in C# and F#

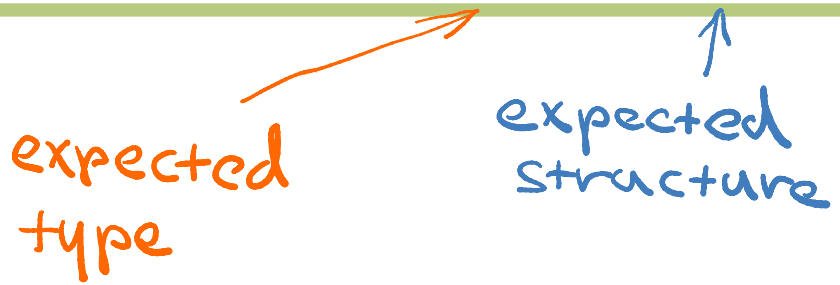
- Data have some structure

  Not understood by the compiler!

- Structure specified locally "as needed"

  Example: Working with XML data

```
MyXmlElement element = GetElement();
int size = element.Attribute<int>("Size");
```



Demo:
Using WorldBank data in C#

Dynamic type in C#

- Operations resolved at runtime

  Result is dynamic, but can be converted

```
dynamic wb = new WorldBank();
dynamic regions = wb.Region(new { PerPage = 100 });
```

```
class WorldBank : DynamicObject {
  public override bool TryInvokeMember(InvokeMemberBinder binder,
      object[] args, out object result)  {
    result = /* member access for 'binder.Name' */
    return true;
  }
}
```

Dynamic in C# and F#

- Dynamic type in C#

```
dynamic element = GetElement();
int size = element.Size;
```

required type

dynamic member

- Dynamic operator in F#

```
let element = GetElement();
let size = element?Size;
```

infered from context

dynamic operator

Simple database access in F#

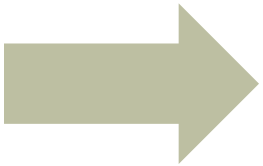
- Operation resolved at compile-time
  - Result has statically known type (not dynamic)
  - Can be inferred from context

```
let getCategories() : seq<int * string> =
  [ for row in db.Query?GetProducts() ->
      row?ID, row?Name ]
```

```
let (?) (x:Row) (name:string) : 'R =
  x.Reader.[name] :?> 'R
```

given by caller

Demo:
Calling database in F#



- **Expression scale**
  - Using dynamic type or operator
- **Program scale**
  - Defining structure in the language
- **Internet scale**
  - Generate structure using type providers

Structure in the language

Describe the target structure

Add hints for mapping

Coerce data to structure

Domain modeling in LINQ

- Data structure described as C# class

```
public partial class Books {
    [Column(Storage="_ID", DbType="Int", IsPrimaryKey=true)]
    public int ID { get; set; }
    [Column(Storage="_Title", DbType="VarChar(100)")]
    public string Title { get; set; }
}
```

- Match data to the structure

  Done at run-time & can fail

```
var q = from p in db.GetTable<Books>()
        select p.Title;
```

Domain modeling in F#

- Simple way to think about data
  - Compose data using simple constructors

Primitive values	Single information (int, string, date)
Records	Combines fixed number of other values
Discriminated unions	Represents one of several options

Demo:
Nicer database access

Nicer database access in F#

- Describe structure using F# types

```
type Book = { ID : int; Title : string }
```

- Match data to the structure (dynamically)

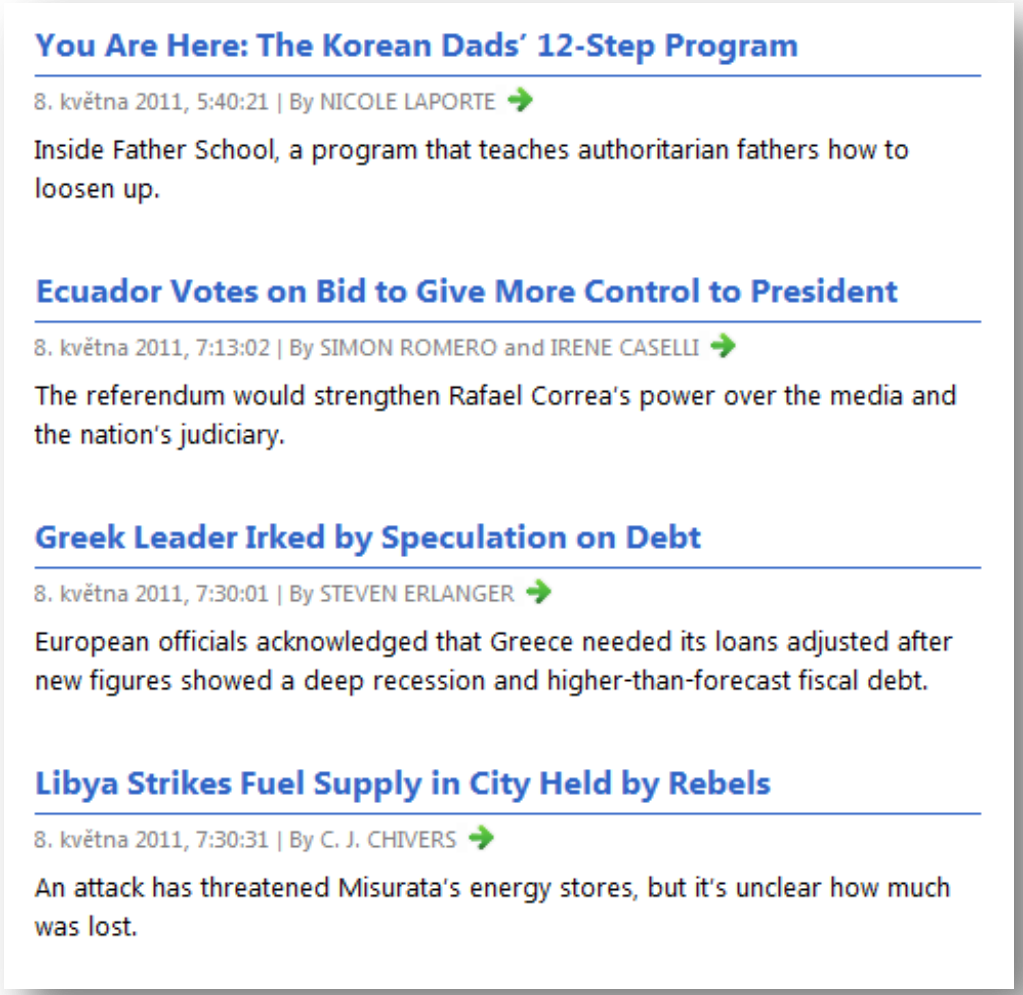
  Type inferred from the context

```
let load() : seq<Book> =
    let db = new DynamicDatabase(connectionString)
    db.Query?GetBooks()
```

type hint

fit data to structure

Domain modeling in F#



You Are Here: The Korean Dads' 12-Step Program

8. května 2011, 5:40:21 | By NICOLE LAPORTE

Inside Father School, a program that teaches authoritarian fathers how to loosen up.

Ecuador Votes on Bid to Give More Control to President

8. května 2011, 7:13:02 | By SIMON ROMERO and IRENE CASELLI

The referendum would strengthen Rafael Correa's power over the media and the nation's judiciary.

Greek Leader Irked by Speculation on Debt

8. května 2011, 7:30:01 | By STEVEN ERLANGER

European officials acknowledged that Greece needed its loans adjusted after new figures showed a deep recession and higher-than-forecast fiscal debt.

Libya Strikes Fuel Supply in City Held by Rebels

8. května 2011, 7:30:31 | By C. J. CHIVERS

An attack has threatened Misurata's energy stores, but it's unclear how much was lost.

Domain modeling in F#

- Define RSS feed structure in F#

```
type Title = Title of string
type Link = Link of string
type Description = Description of string

/// Item consists of title, link and description
type Item =
   Item of Title * Link * Description

/// Represents channel with title (etc.) and list of items
type Channel =
   Channel of Title * Link * Description * list<Item>

/// Represents RSS feed containing a channel
type Rss = Rss of Channel
```

Working with XML

- Loading RSS feed

gets infered type

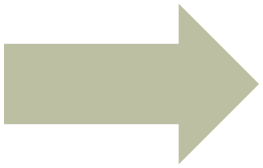
type inference

```
let rss = StructuralXml.Load("http://.../feed")

match rss.Root with
| Rss(Channel(Title title, _, _, items)) ->
    printfn "%s (%d items)" title items.Length
```

- Key components
  - Domain model in the language
  - Library for matching data to the model
  - Can fail if model is not in sync

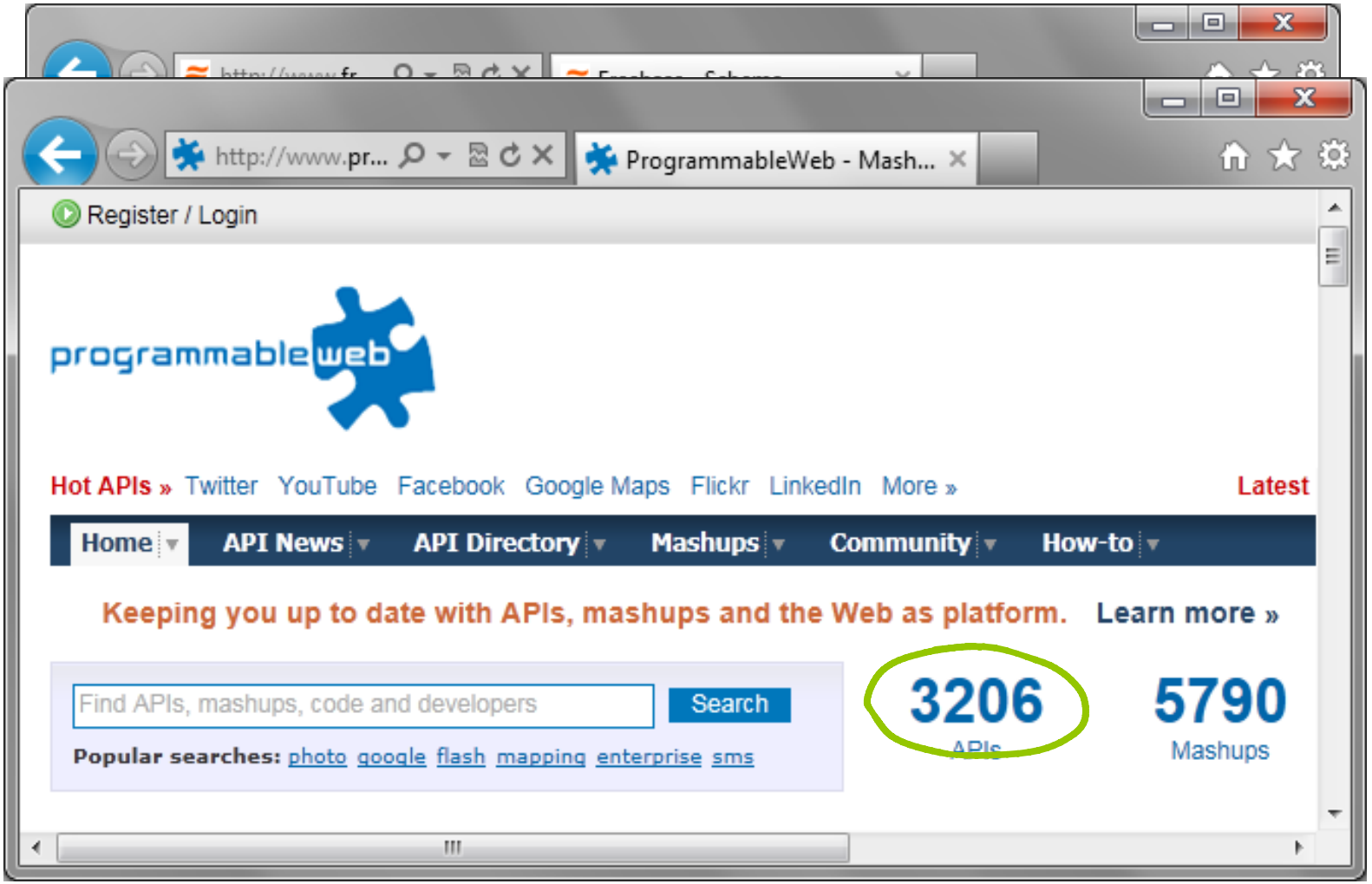
Demo:
Working with XML



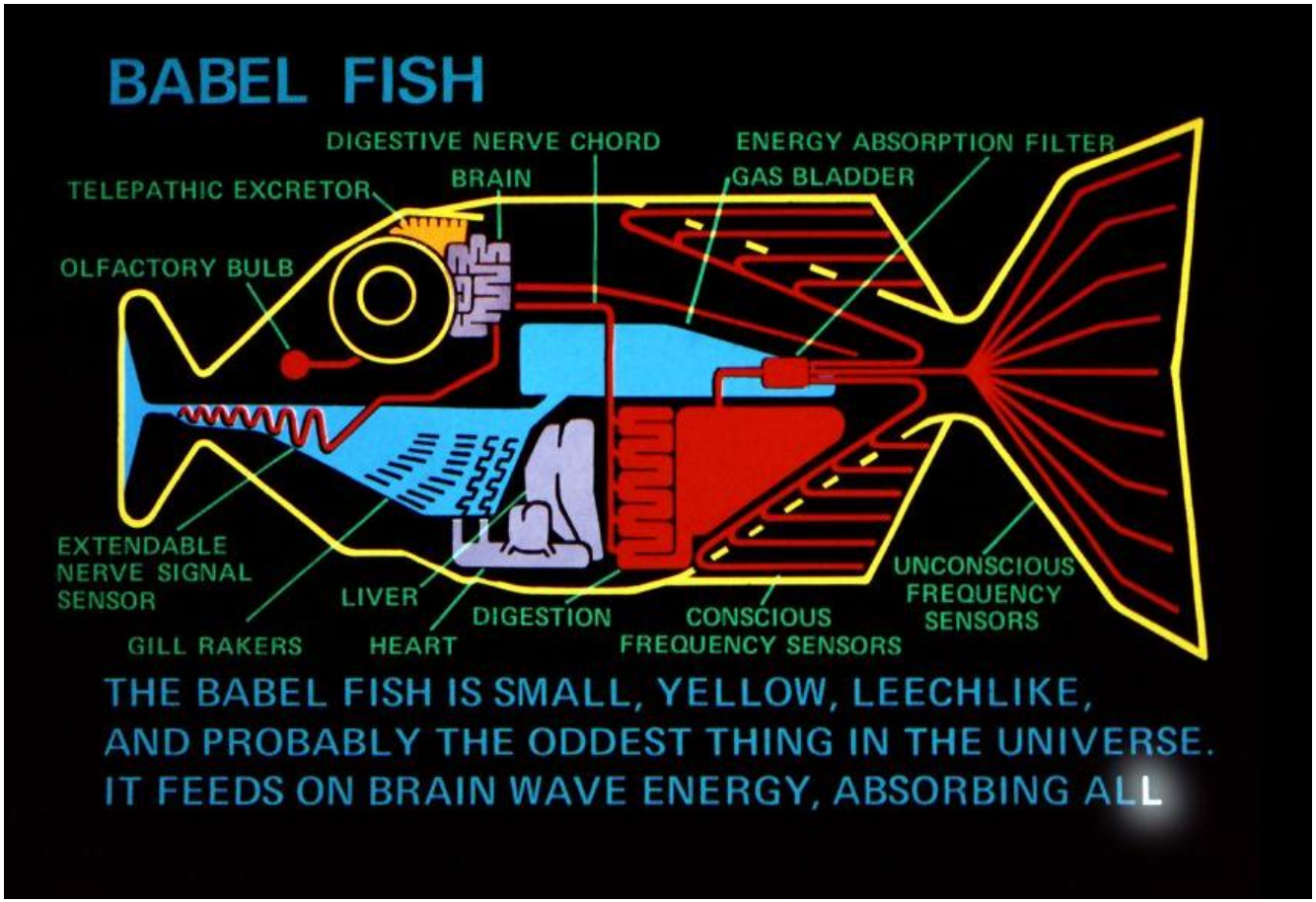
- **Expression scale**
  - Using dynamic type or operator
- **Program scale**
  - Defining structure in the language
- **Internet scale**
  - Generate structure using type providers

The Problem

Defining structure in the language doesn't scale!



The Solution



Quick, stick this fish in your compiler!

Demo:
Accessing WorldBank data

What is a type provider?

- Assembly containing a special type

```
public interface ITypeProvider {
   Type[] GetTypes();

   Expression GetInvokerExpression
      ( MethodBase method,
        ParameterExpression[] params );

   event EventHandler Invalidate;
}
```

- Creating them is easier than it looks

  Method calls replaced by expressions

  …or you can generate real types

Type provider trickery

- Gives you a new way of thinking
  - This wasn't really done before…
- Delegates many questions to the provider
  - Different views? Different versions?
  - What if data source is not available?
- F# is a great playground
  - Object-oriented concepts help
  - More information with units of measure

Demo:
Accessing Freebase data

Summary

Building a bridge between two structures

Dynamic type or operator

- Explicit
- Syntax in language
- Local scale (expression)

Domain model

- Explicit
- Classes or data types
- Program or library scale

Type providers

- Implicit
- Generated object types
- Web scale

Discussion

Questions & Answers?

- F# training at SkillsMatter
  - **Functional Programming with .NET** (27-28 October)
  - **Real-World F# Programming** (15-16 December)
- Contact
  - http://tomasp.net | @tomaspetricek | tomas@tomasp.net

BONUS

World Bank provider details

- At runtime countries & indicators are strings
  - For example "GBR" or "GC.DOD.TOTL.GD.ZS"
  - Members translated to runtime functions

```
type Runtime =
  static member GetValuesByCountryAndIndicator
      (country:string, indicator:string) : seq<int * float> =
    seq { for (k, v) in WorldBank.GetData [country; indicator] do
            if not (String.IsNullOrEmpty v) then
              yield int k, float v }

 static member GetCountriesByRegion(code:string) : seq<string> =
    seq { for (key, _) in WorldBank.GetCountries(code) -> key }
```

Calling PHP from C#

- Dynamically typed PHP object

```
class SampleObj {
  function Add($a, $b) { return $a + $b; }
}
```

- Required structure specified as a C# interface

```
[DuckType]
public interface ISampleObj {
  int Add(int i1, int i2);
}
```

```
ISampleObj so = ctx.New<ISampleObj>("SampleObj");
int res = so.Add(18, 26);
```