F# Type Providers

"Statically Typed Language Support for Internet-scale Information Spaces"

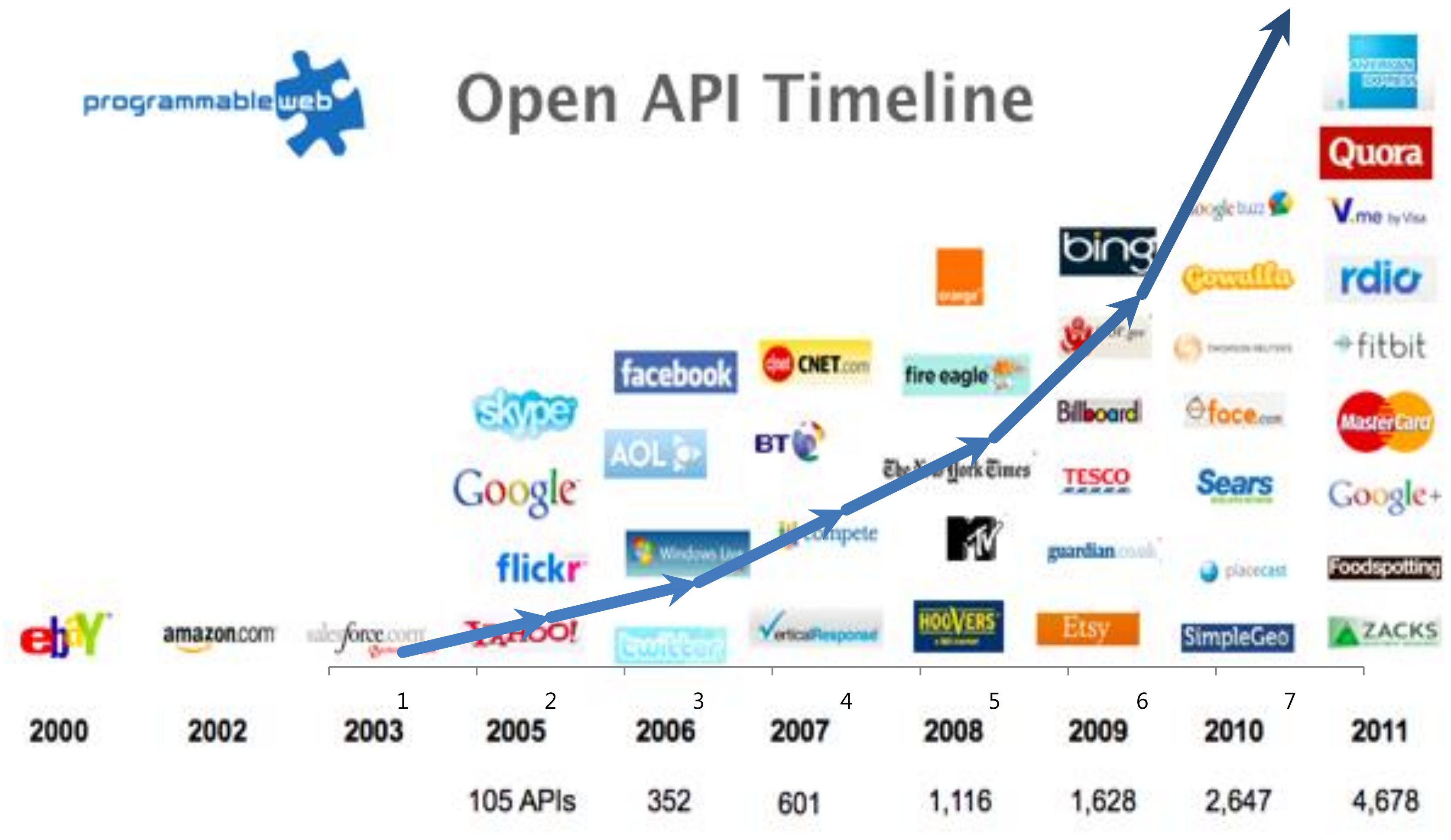
Overview + Exotica

Don Syme, F# Community Contributor, @dsyme

In conjunction with many others in F# Community

Proposition 1
The world is information-rich

The Information Revolution



Proposition 2
Modern programming is intensely information-rich

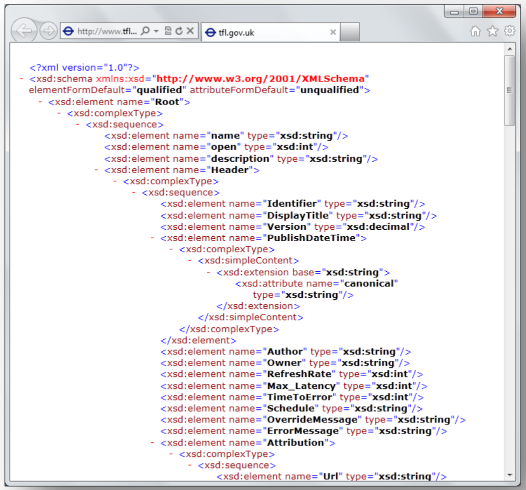
Proposition 3
Our languages are information-sparse

Proposition 4
This is a problem

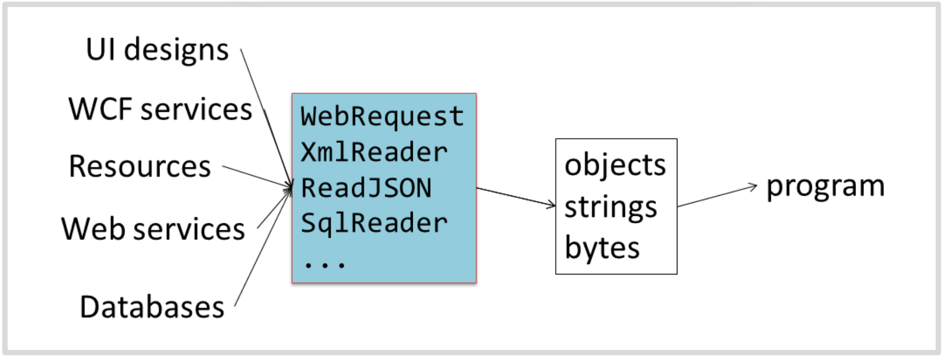
(especially for strongly-typed languages)

The developer’s perspective

- Languages do not integrate information
  - Non-intuitive
  - Not simple
  - Disorganised
  - Static
  - High friction



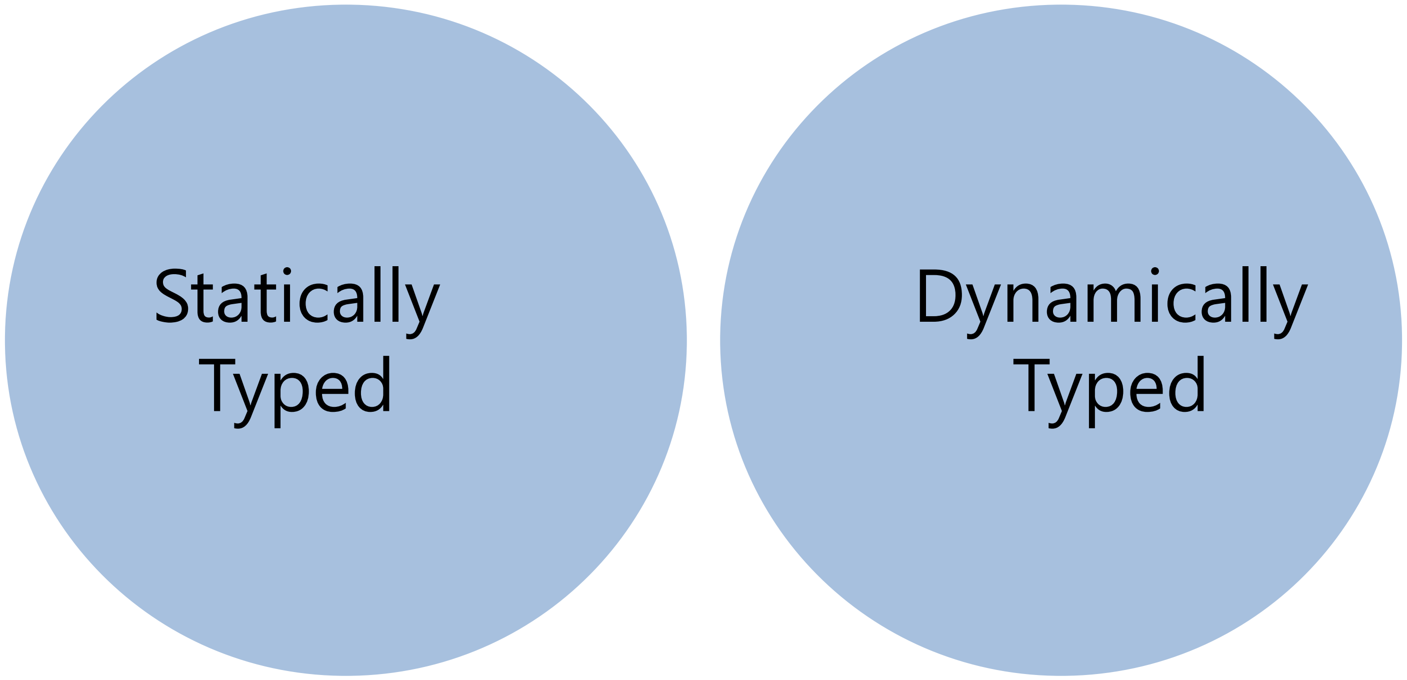
```
  <?xml version="1.0"?>
- <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified" attributeFormDefault="unqualified">
  - <xsd:element name="Root">
    - <xsd:complexType>
      - <xsd:sequence>
          <xsd:element name="name" type="xsd:string"/>
          <xsd:element name="open" type="xsd:int"/>
          <xsd:element name="description" type="xsd:string"/>
        - <xsd:element name="Header">
          - <xsd:complexType>
            - <xsd:sequence>
                <xsd:element name="Identifier" type="xsd:string"/>
                <xsd:element name="DisplayTitle" type="xsd:string"/>
                <xsd:element name="Version" type="xsd:decimal"/>
              - <xsd:element name="PublishDateTime">
                - <xsd:complexType>
                  - <xsd:simpleContent>
                    - <xsd:extension base="xsd:string">
                        <xsd:attribute name="canonical"
                          type="xsd:string"/>
                      </xsd:extension>
                    </xsd:simpleContent>
                  </xsd:complexType>
                </xsd:element>
                <xsd:element name="Author" type="xsd:string"/>
                <xsd:element name="Owner" type="xsd:string"/>
                <xsd:element name="RefreshRate" type="xsd:int"/>
                <xsd:element name="Max_Latency" type="xsd:int"/>
                <xsd:element name="TimeToError" type="xsd:int"/>
                <xsd:element name="Schedule" type="xsd:string"/>
                <xsd:element name="OverrideMessage" type="xsd:string"/>
                <xsd:element name="ErrorMessage" type="xsd:string"/>
              - <xsd:element name="Attribution">
                - <xsd:complexType>
                  - <xsd:sequence>
                      <xsd:element name="Url" type="xsd:string"/>
```



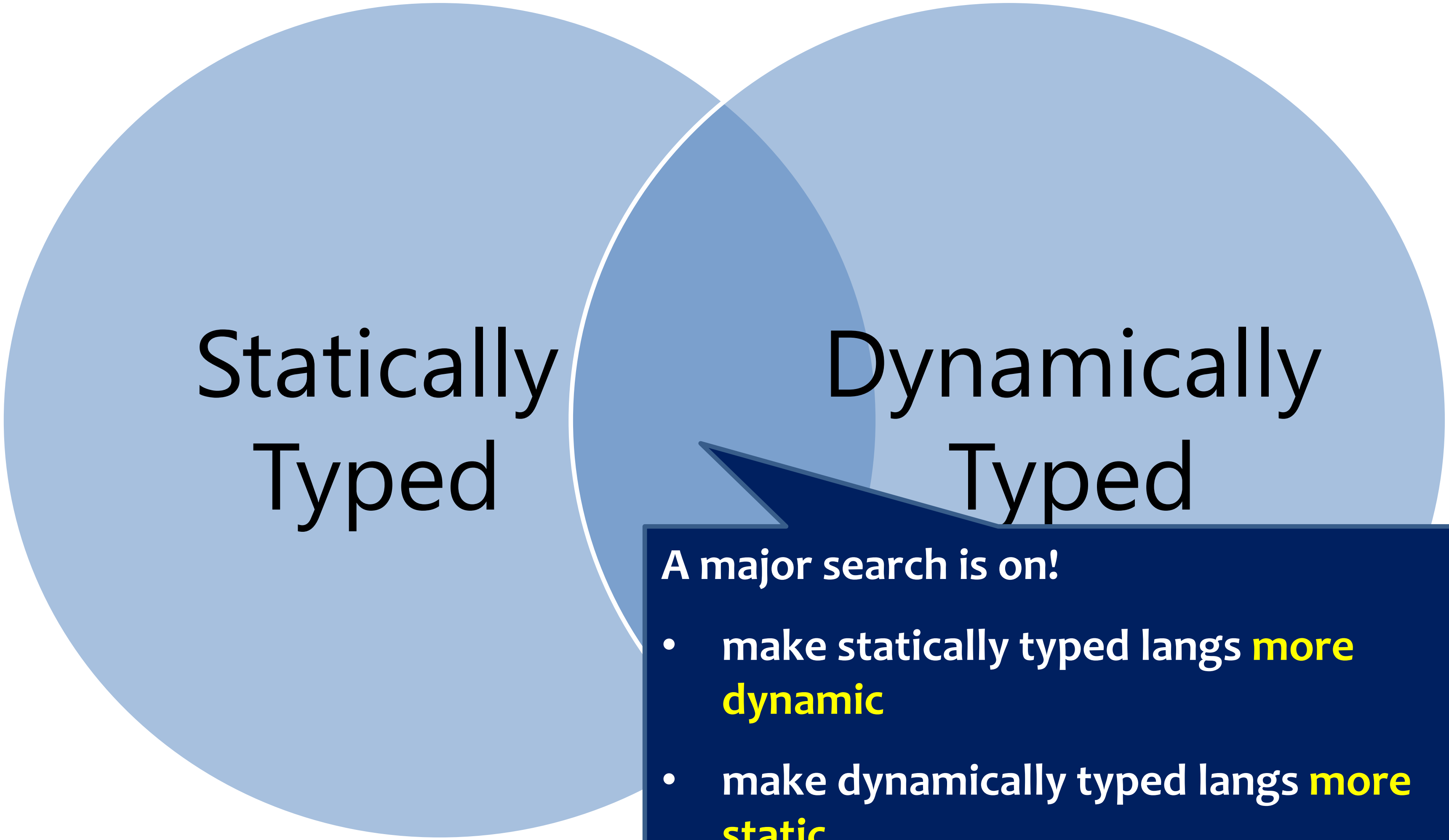
We need to bring information **into** the language…

At **internet-scale, strongly tooled, strongly typed**

Paradigm Locator



Paradigm Locator



A major search is on!

- make statically typed langs **more dynamic**
- make dynamically typed langs **more static**
- **apply modertated static typing** much more broadly

This R&D is done in the context of the open language F#

Lots of good reasons for that, see our tech report "Strongly Typed Language Support for Internet-Scale Information Spaces"

A bit about F#...

F# is Open Source
F# is Cross Platform

fsharp.org
fsharp.org/use/mac
fsharp.org/use/linux
fsharp.org/use/android
fsharp.org/use/ios
fsharp.org/use/windows
fsharp.org/use/freebsd

F# is changing…

"F# is for Windows" → F# runs on many platforms

F# is changing…

“Microsoft makes F#”

F# has many contributors

F# is changing…

One perspective (Microsoft's)
http://msdn.microsoft.com

Many perspectives
http://fsharp.org

Learn more

fsharp.org

fsharp.org/testimonials
fsharp.org/teaching/research.html

Back to the main topic…

We need to bring information **into** the language…

At **internet-scale, strongly tooled, strongly typed**

Demo: Integrate all of freebase.com

"as if it were a library"

40M entities, 1Billion facts, 24,000 types, 65,000 properties

Demo: F# to WorldBank

A Type Provider is….

"Just like a library"

"A design-time component that computes a space of types and methods on-demand…"

"An adaptor between data/services and the .NET type system…"

"On-demand, scalable compile-time provision of type/module definitions…"

Theme #1

On-Demand Type Environment = Internet Scalable

On-Demand Type Provision

```
let data = Freebase.GetDataContext()
```

1. Compiler/IDE requests metadata for symbol `GetDataContext`
   - ✓ Provider reports return type of `FreebaseDataContext`

```
data.
```

2. Compiler/IDE requests contents of type `FreebaseDataContext`
   - ✓ Provider asks Freebase metadata service for top-level domains
   - ✓ Provider reports top-level domains of Freebase as properties of the type

```
data.Society
```

3. Compiler/IDE requests metadata for symbol `Society`

…

Theme #2

Many Data Sources, One Mechanism

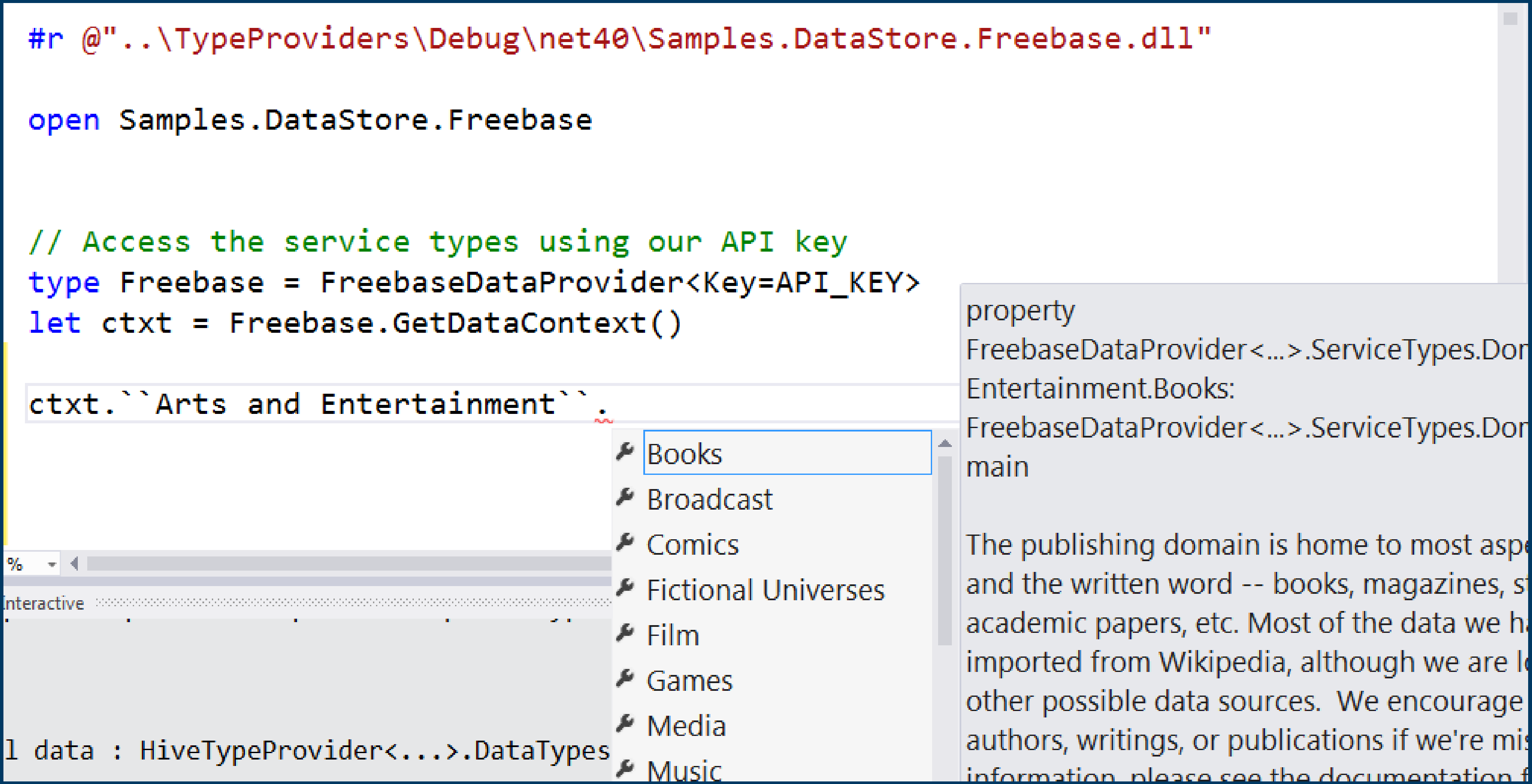
Note: Language still contains no data

Open architecture

You can write your own type provider

All your types are belong to us….

Freebase



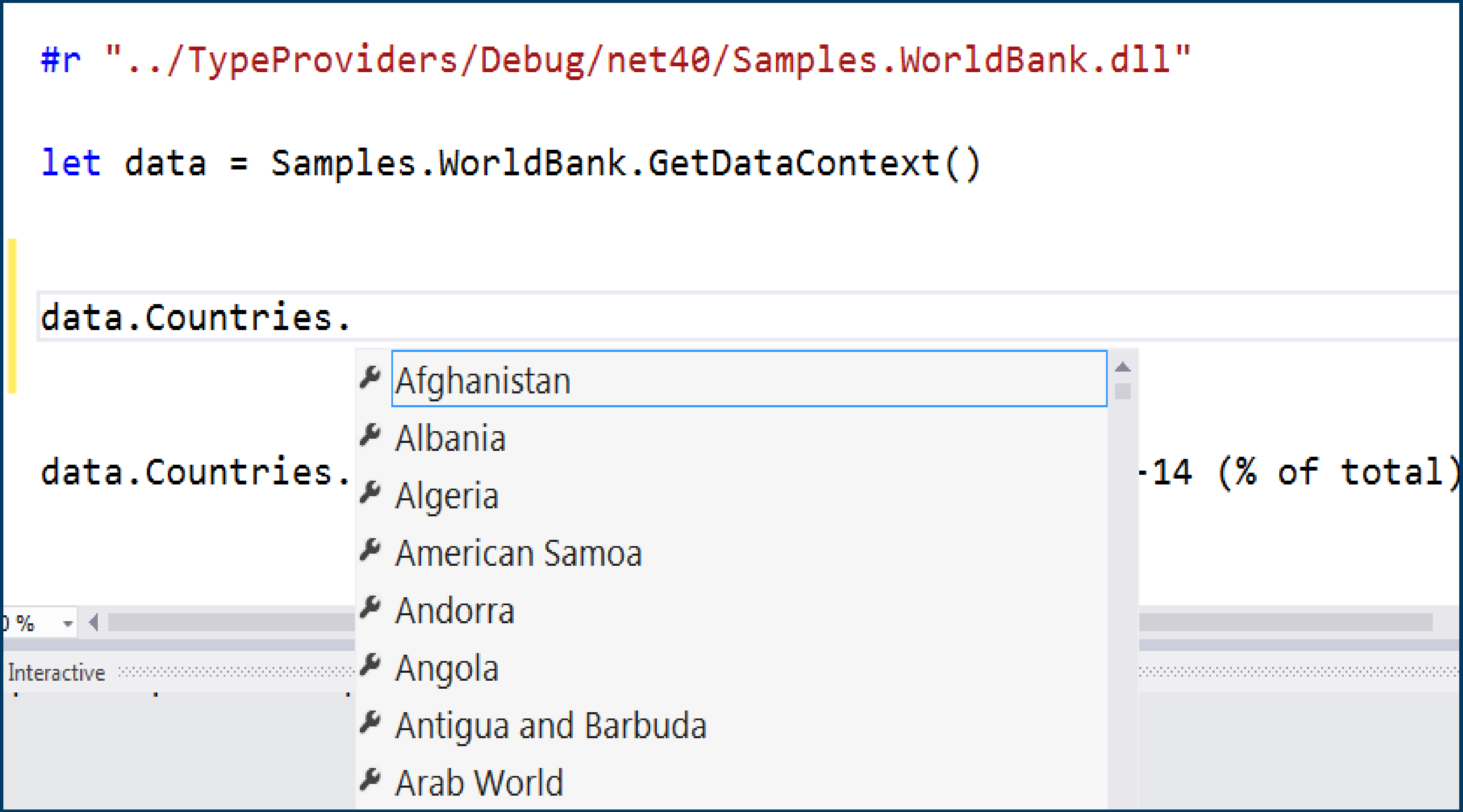
```
#r @"..\TypeProviders\Debug\net40\Samples.DataStore.Freebase.dll"

open Samples.DataStore.Freebase


// Access the service types using our API key
type Freebase = FreebaseDataProvider<Key=API_KEY>
let ctxt = Freebase.GetDataContext()

ctxt.``Arts and Entertainment``.
```

property
FreebaseDataProvider<...>.ServiceTypes.Dor
Entertainment.Books:
FreebaseDataProvider<...>.ServiceTypes.Dor
main

The publishing domain is home to most aspe
and the written word -- books, magazines, s
academic papers, etc. Most of the data we h
imported from Wikipedia, although we are l
other possible data sources. We encourage
authors, writings, or publications if we're mi
information, please see the documentation f

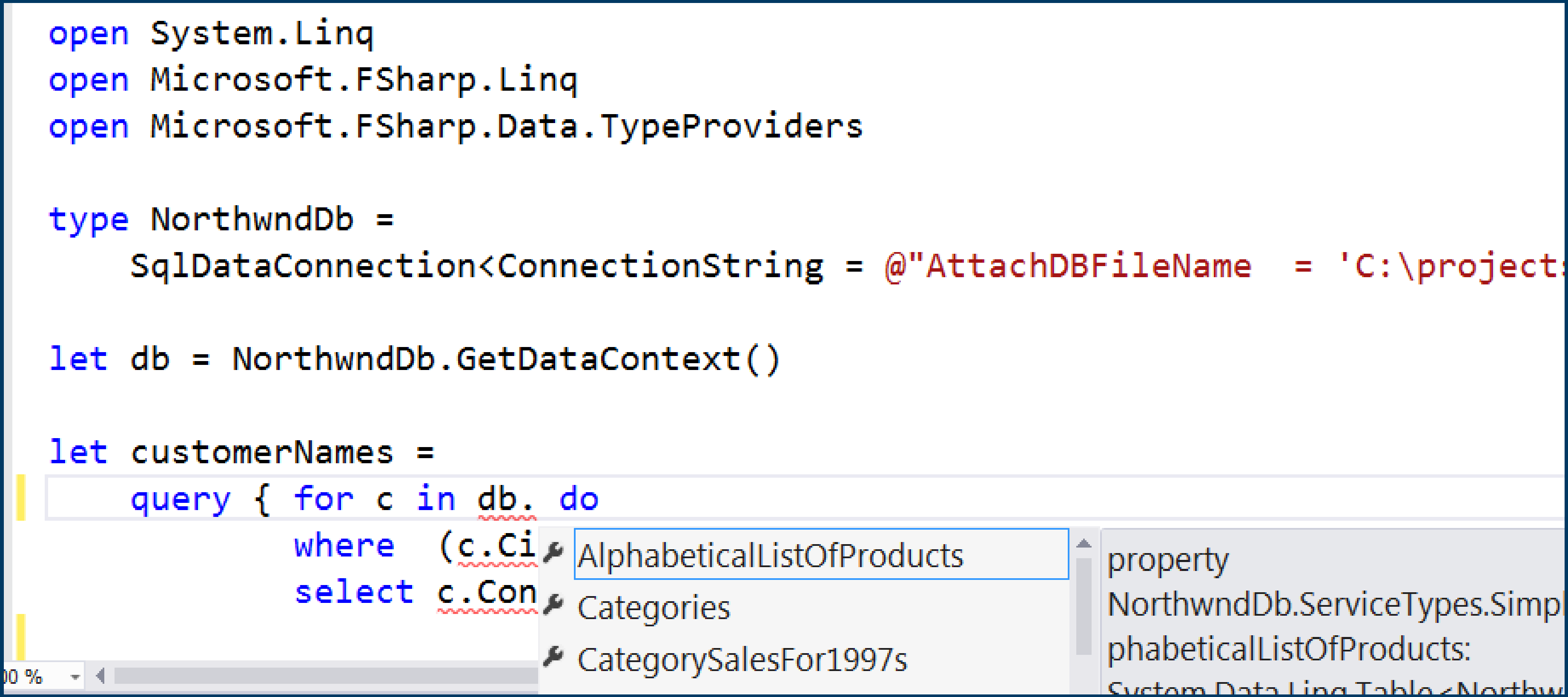
```
l data : HiveTypeProvider<...>.DataTypes
```

World Bank



```
#r "../TypeProviders/Debug/net40/Samples.WorldBank.dll"

let data = Samples.WorldBank.GetDataContext()


data.Countries.


data.Countries.                                                  -14 (% of total)
```

SQL

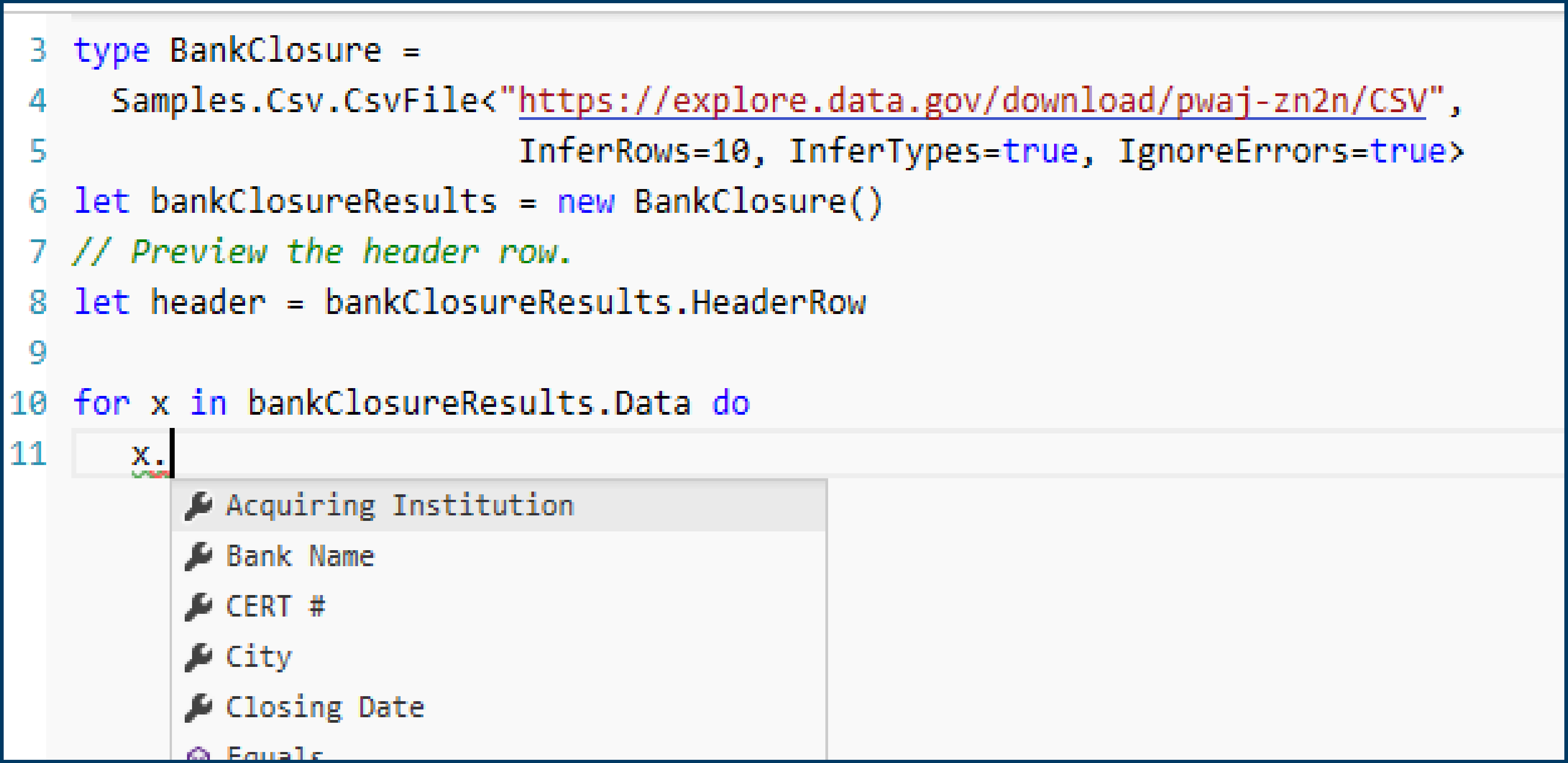


```
open System.Linq
open Microsoft.FSharp.Linq
open Microsoft.FSharp.Data.TypeProviders

type NorthwndDb =
    SqlDataConnection<ConnectionString = @"AttachDBFileName  = 'C:\project

let db = NorthwndDb.GetDataContext()

let customerNames =
    query { for c in db. do
                where  (c.Ci
                select c.Con
```

AlphabeticalListOfProducts
Categories
CategorySalesFor1997s

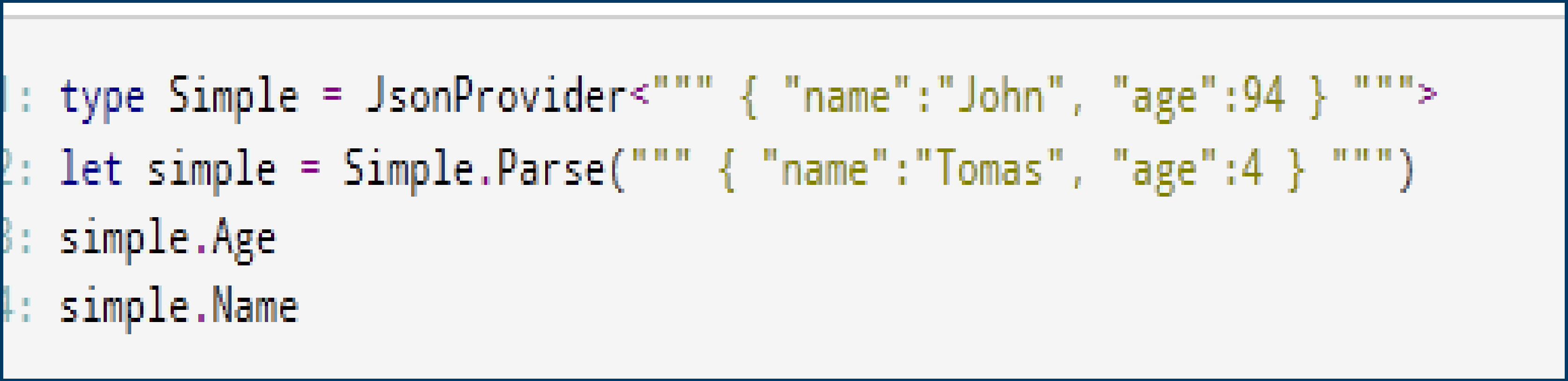
property
NorthwndDb.ServiceTypes.Simpl
phabeticalListOfProducts:
System.Data.Linq.Table<Northw

CSV



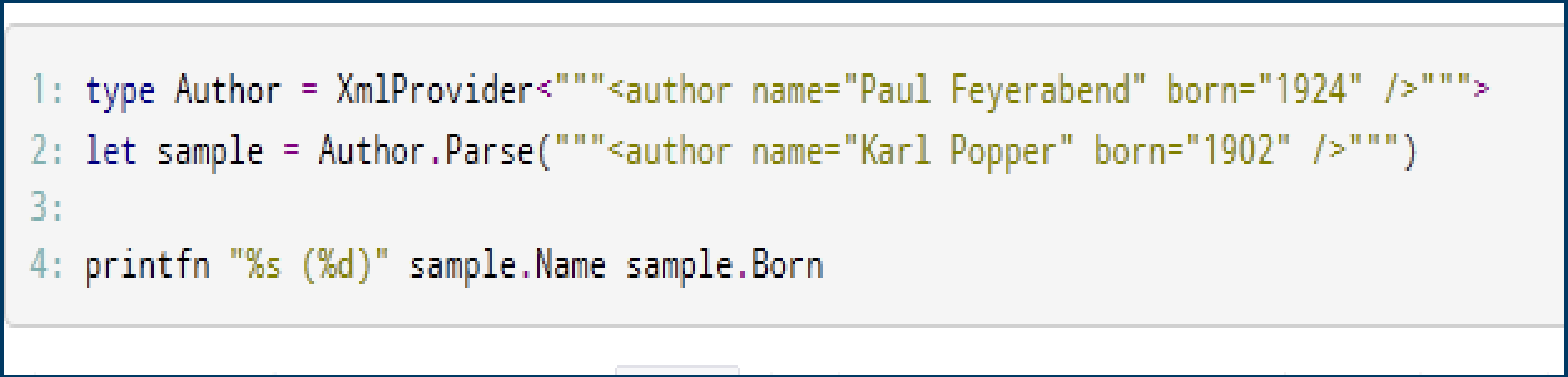
```
type BankClosure =
  Samples.Csv.CsvFile<"https://explore.data.gov/download/pwaj-zn2n/CSV",
                      InferRows=10, InferTypes=true, IgnoreErrors=true>
let bankClosureResults = new BankClosure()
// Preview the header row.
let header = bankClosureResults.HeaderRow

for x in bankClosureResults.Data do
    x.
```

JSON



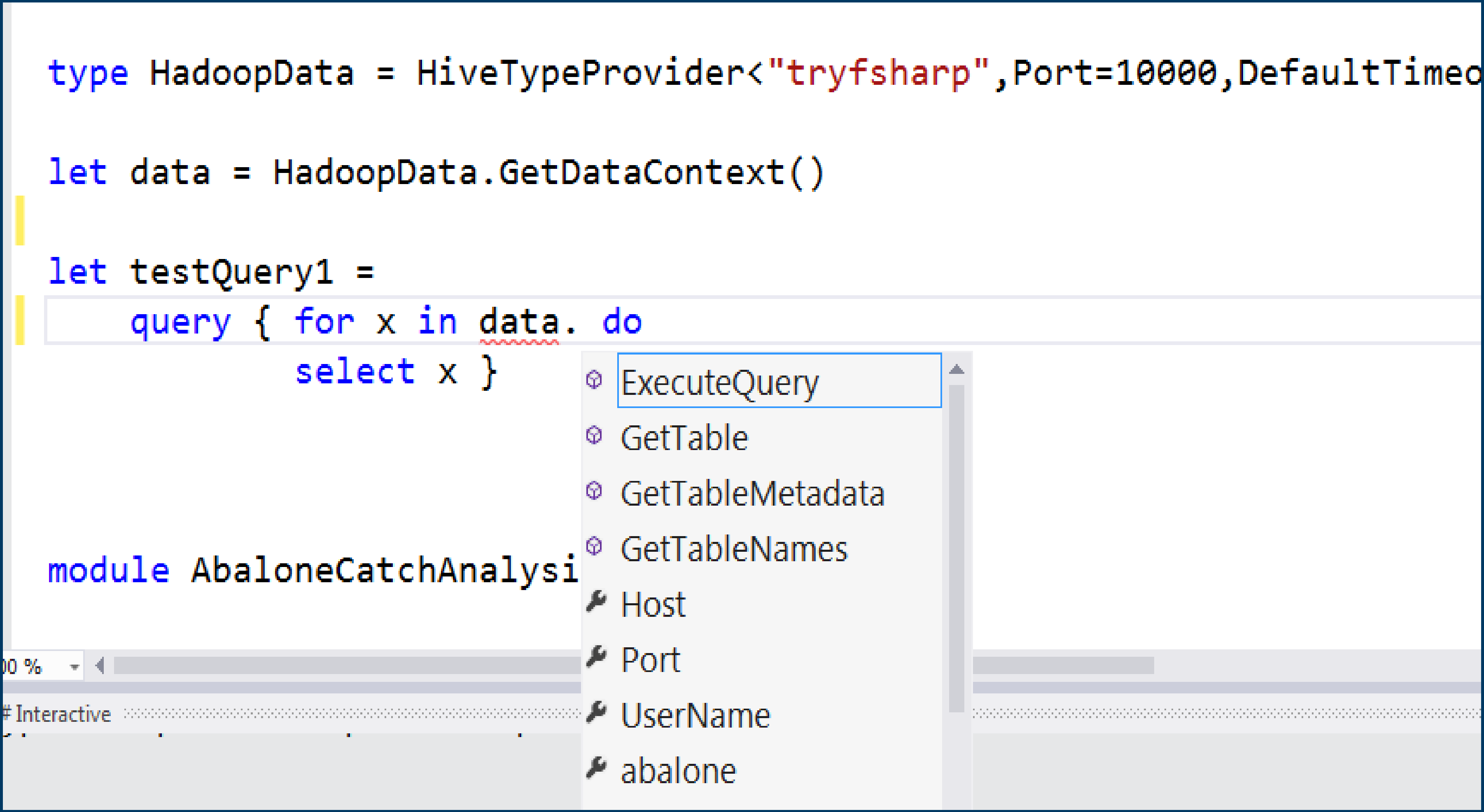
```
type Simple = JsonProvider<""" { "name":"John", "age":94 } """>
let simple = Simple.Parse(""" { "name":"Tomas", "age":4 } """)
simple.Age
simple.Name
```

XML



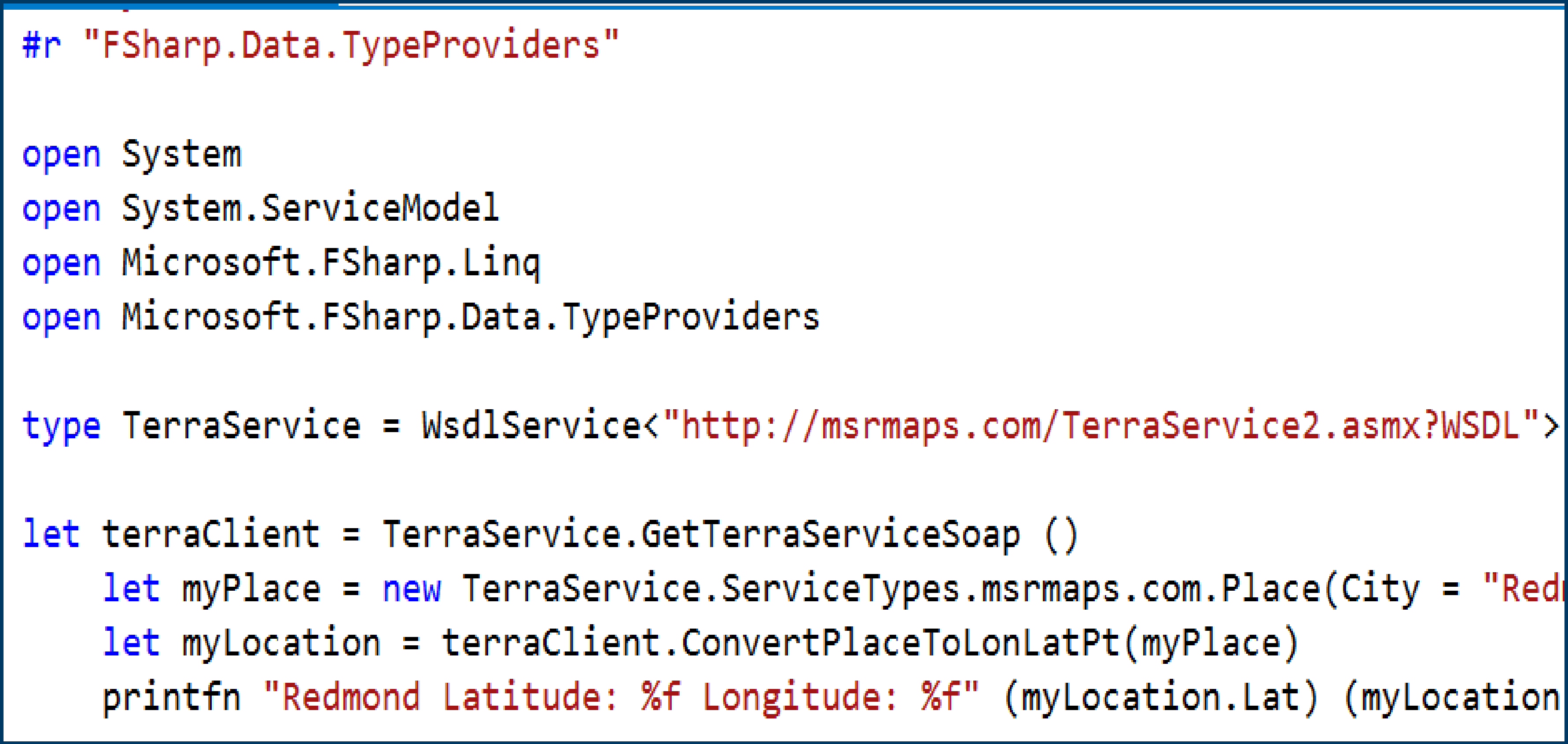
```
type Author = XmlProvider<"""<author name="Paul Feyerabend" born="1924" />""">
let sample = Author.Parse("""<author name="Karl Popper" born="1902" />""")

printfn "%s (%d)" sample.Name sample.Born
```

Hadoop/Hive



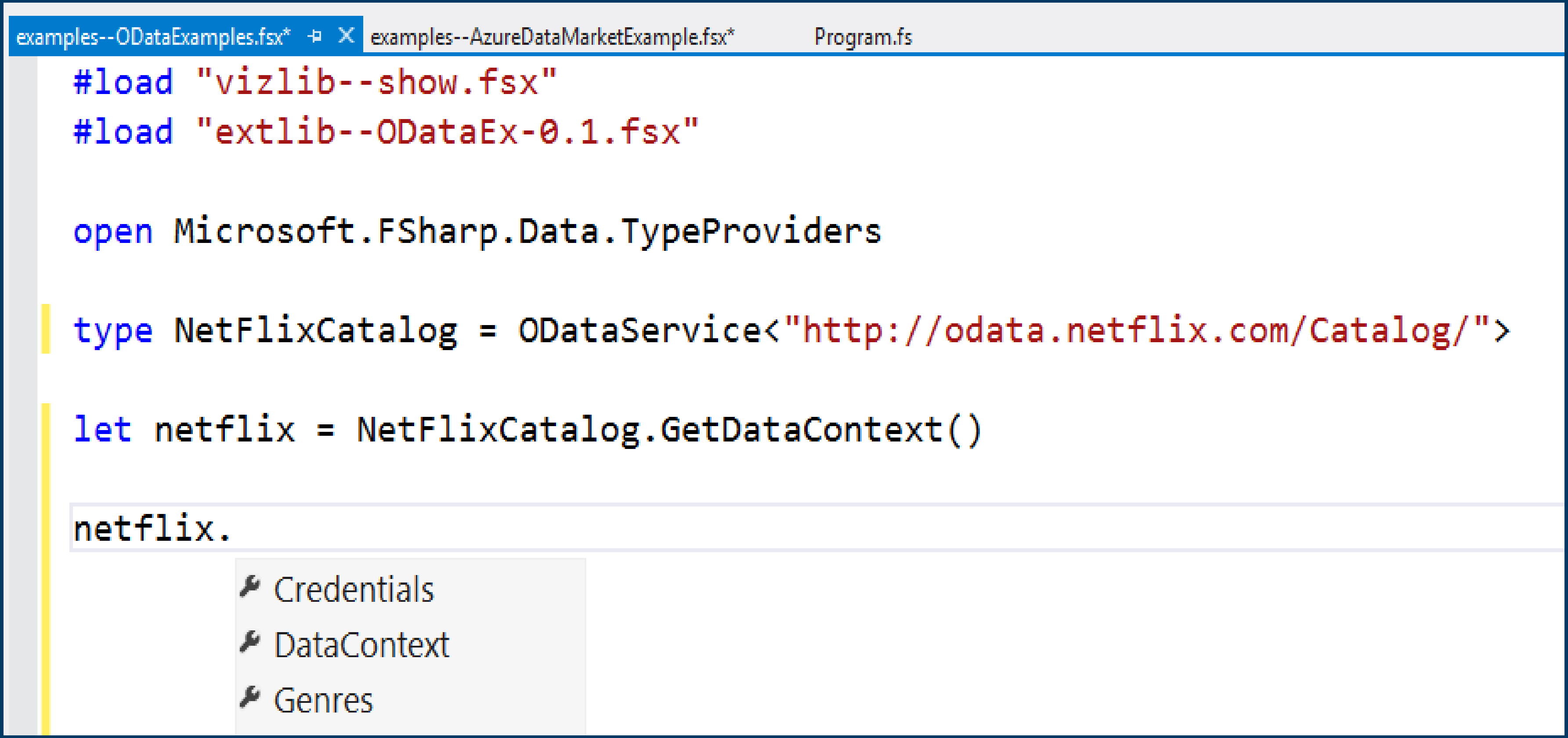
```
type HadoopData = HiveTypeProvider<"tryfsharp",Port=10000,DefaultTimeo

let data = HadoopData.GetDataContext()

let testQuery1 =
    query { for x in data. do
                select x }



module AbaloneCatchAnalysi
```

WSDL



```
#r "FSharp.Data.TypeProviders"

open System
open System.ServiceModel
open Microsoft.FSharp.Linq
open Microsoft.FSharp.Data.TypeProviders

type TerraService = WsdlService<"http://msrmaps.com/TerraService2.asmx?WSDL">

let terraClient = TerraService.GetTerraServiceSoap ()
    let myPlace = new TerraService.ServiceTypes.msrmaps.com.Place(City = "Red
    let myLocation = terraClient.ConvertPlaceToLonLatPt(myPlace)
    printfn "Redmond Latitude: %f Longitude: %f" (myLocation.Lat) (myLocation
```

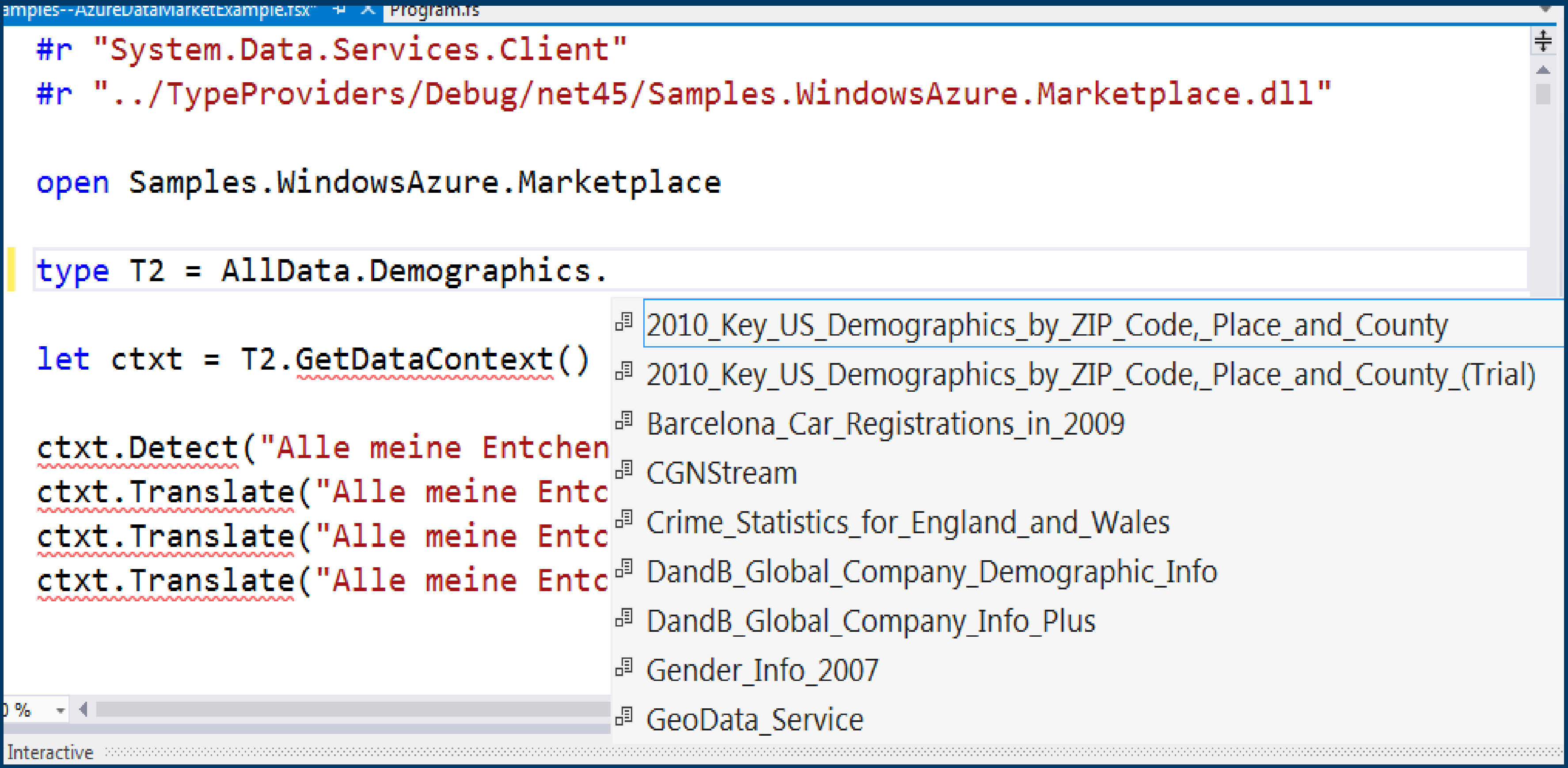
OData


examples--ODataExamples.fsx*
examples--AzureDataMarketExample.fsx*
Program.fs

```
#load "vizlib--show.fsx"
#load "extlib--ODataEx-0.1.fsx"

open Microsoft.FSharp.Data.TypeProviders

type NetFlixCatalog = ODataService<"http://odata.netflix.com/Catalog/">

let netflix = NetFlixCatalog.GetDataContext()

netflix.
```

Credentials
DataContext
Genres

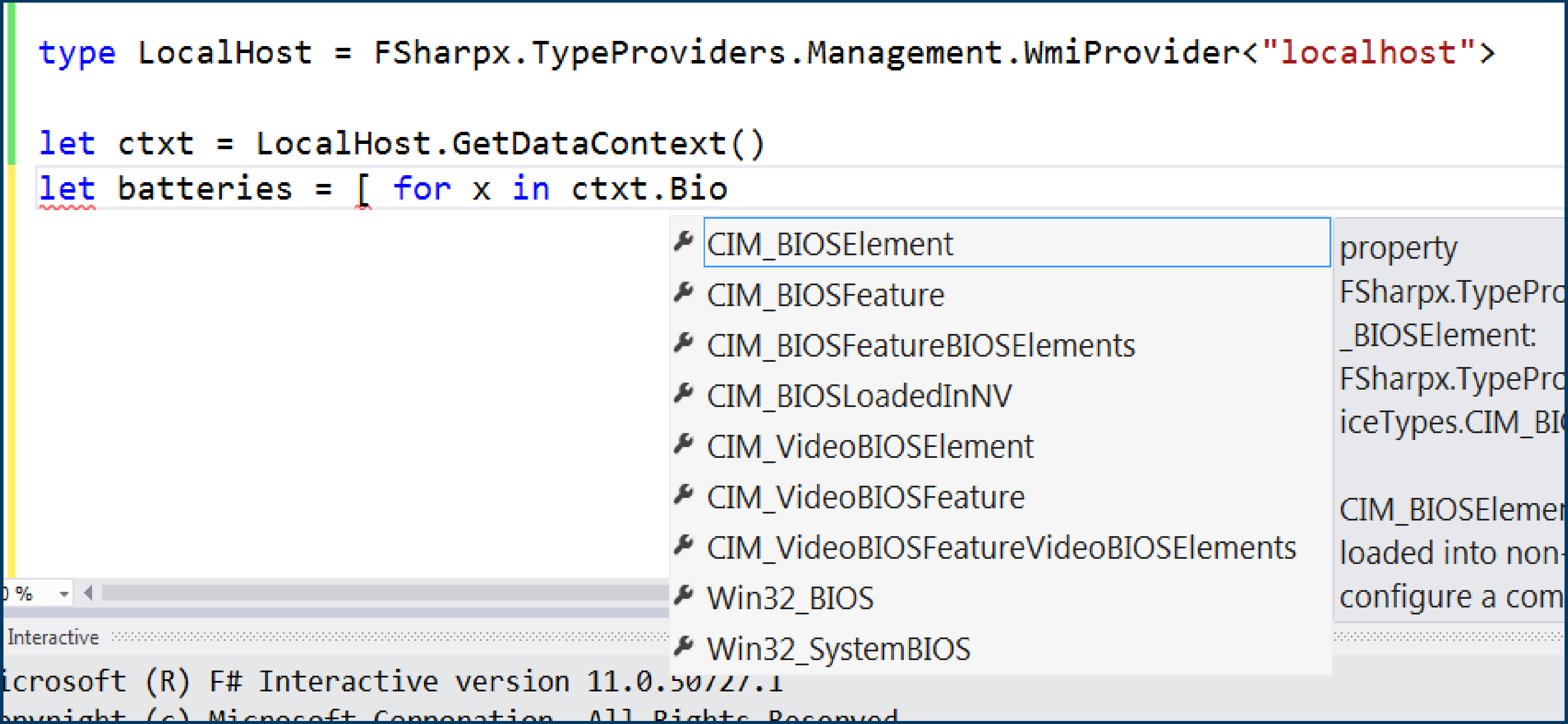
Azure Data Market



```
#r "System.Data.Services.Client"
#r "../TypeProviders/Debug/net45/Samples.WindowsAzure.Marketplace.dll"

open Samples.WindowsAzure.Marketplace

type T2 = AllData.Demographics.

let ctxt = T2.GetDataContext()

ctxt.Detect("Alle meine Entchen
ctxt.Translate("Alle meine Entc
ctxt.Translate("Alle meine Entc
ctxt.Translate("Alle meine Entc
```

```
2010_Key_US_Demographics_by_ZIP_Code,_Place_and_County
2010_Key_US_Demographics_by_ZIP_Code,_Place_and_County_(Trial)
Barcelona_Car_Registrations_in_2009
CGNStream
Crime_Statistics_for_England_and_Wales
DandB_Global_Company_Demographic_Info
DandB_Global_Company_Info_Plus
Gender_Info_2007
GeoData_Service
```

WMI



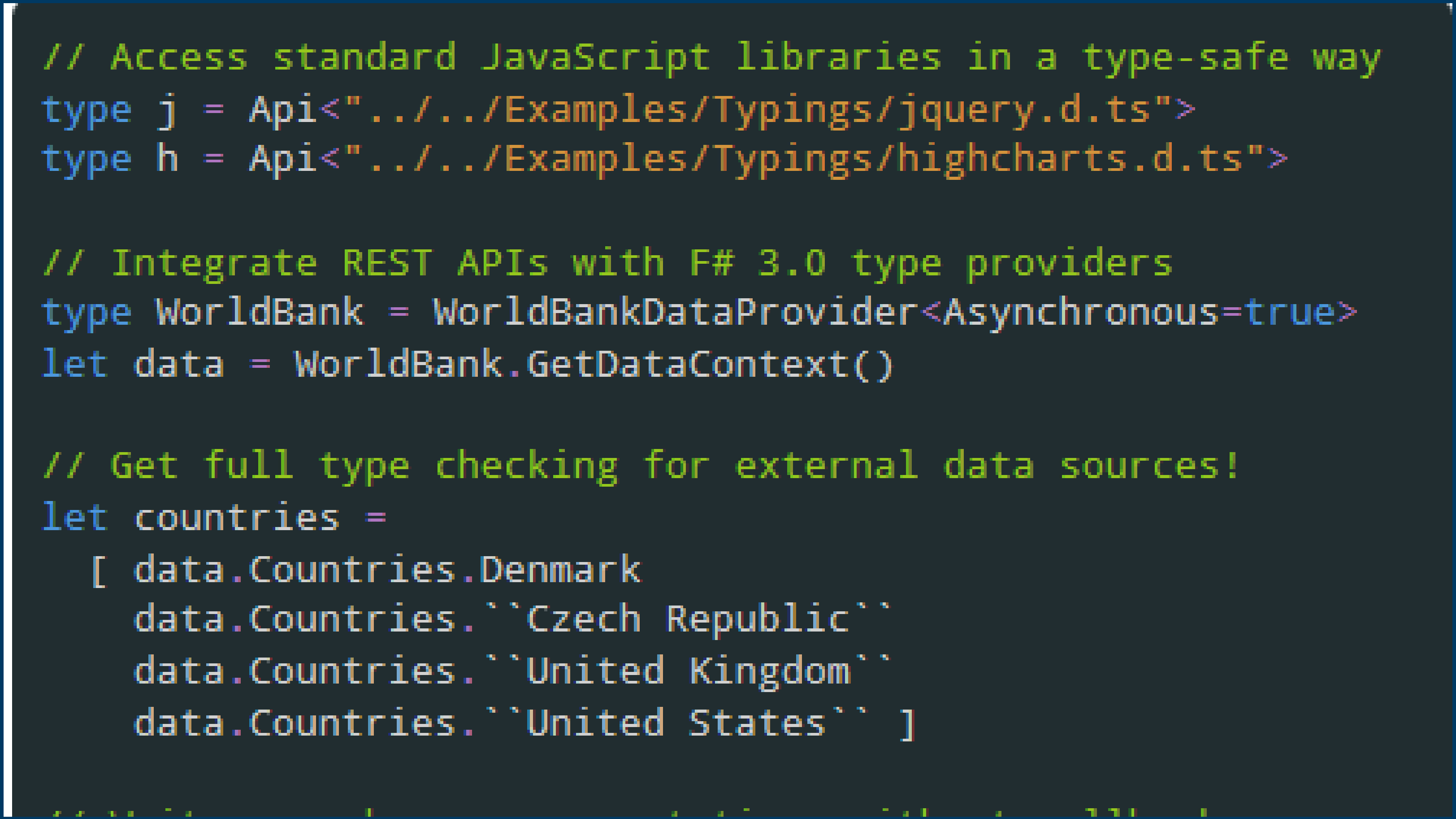
```
type LocalHost = FSharpx.TypeProviders.Management.WmiProvider<"localhost">

let ctxt = LocalHost.GetDataContext()
let batteries = [ for x in ctxt.Bio
```

```
CIM_BIOSElement
CIM_BIOSFeature
CIM_BIOSFeatureBIOSElements
CIM_BIOSLoadedInNV
CIM_VideoBIOSElement
CIM_VideoBIOSFeature
CIM_VideoBIOSFeatureVideoBIOSElements
Win32_BIOS
Win32_SystemBIOS
```

```
property
FSharpx.TypePro
_BIOSElement:
FSharpx.TypePro
iceTypes.CIM_BI

CIM_BIOSElemen
loaded into non-
configure a com
```

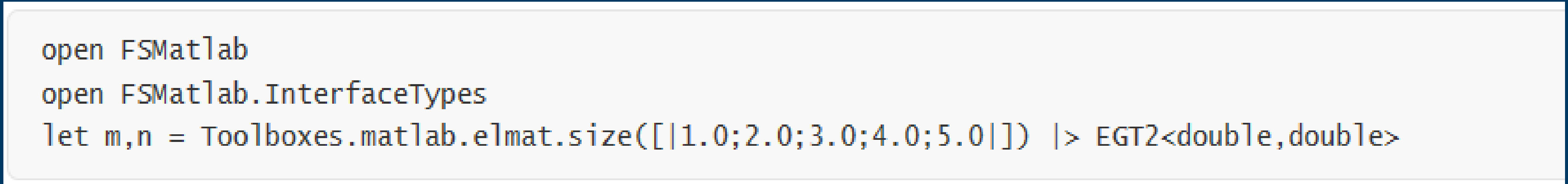
```
icrosoft (R) F# Interactive version 11.0.50727.1
opyright (c) Microsoft Corporation. All Rights Reserved.
```

JavaScript/TypeScript

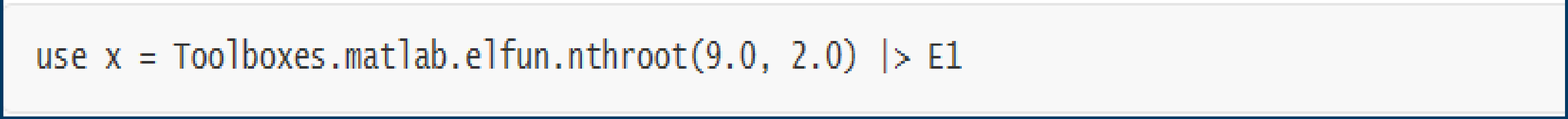


```
// Access standard JavaScript libraries in a type-safe way
type j = Api<"../../Examples/Typings/jquery.d.ts">
type h = Api<"../../Examples/Typings/highcharts.d.ts">

// Integrate REST APIs with F# 3.0 type providers
type WorldBank = WorldBankDataProvider<Asynchronous=true>
let data = WorldBank.GetDataContext()

// Get full type checking for external data sources!
let countries =
  [ data.Countries.Denmark
    data.Countries.``Czech Republic``
    data.Countries.``United Kingdom``
    data.Countries.``United States`` ]
```

Matlab

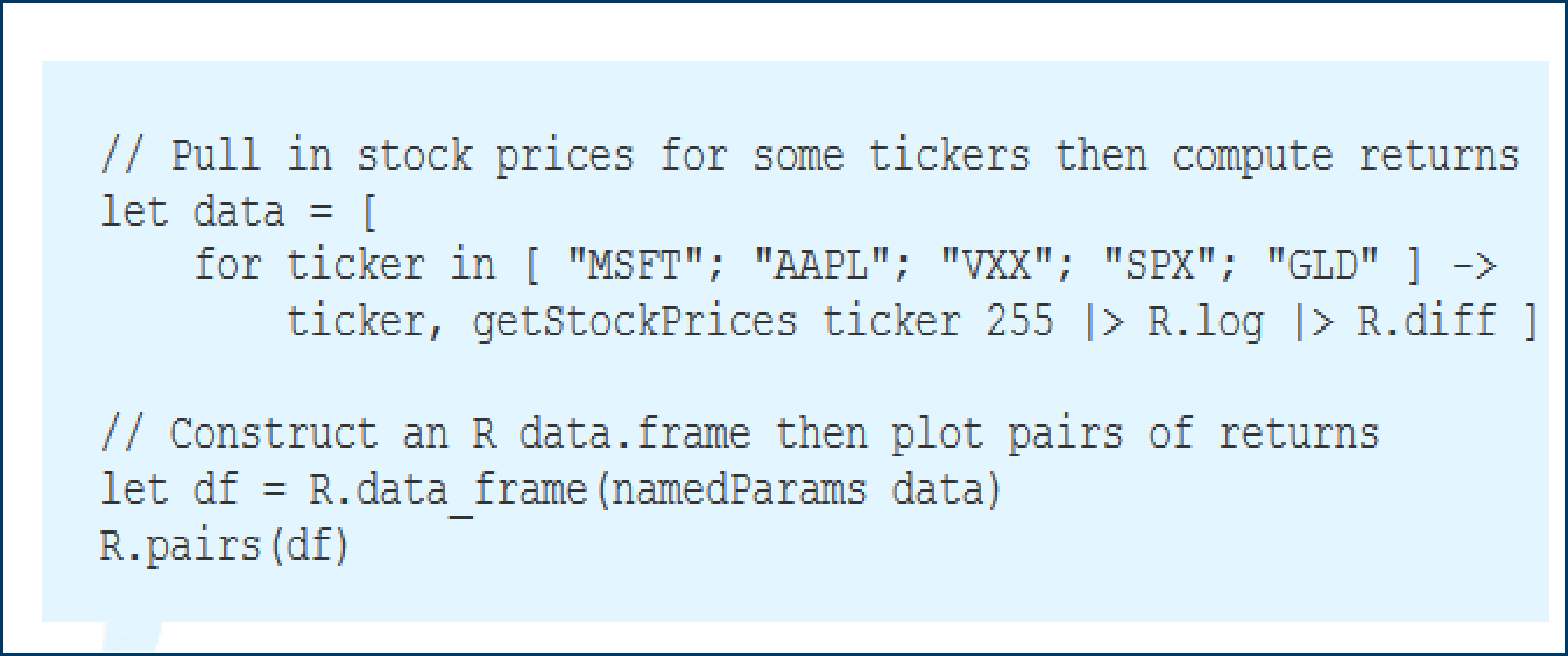


```
open FSMatlab
open FSMatlab.InterfaceTypes
let m,n = Toolboxes.matlab.elmat.size([|1.0;2.0;3.0;4.0;5.0|]) |> EGT2<double,double>
```



```
use x = Toolboxes.matlab.elfun.nthroot(9.0, 2.0) |> E1
```

R



```
// Pull in stock prices for some tickers then compute returns
let data = [
    for ticker in [ "MSFT"; "AAPL"; "VXX"; "SPX"; "GLD" ] ->
        ticker, getStockPrices ticker 255 |> R.log |> R.diff ]

// Construct an R data.frame then plot pairs of returns
let df = R.data_frame(namedParams data)
R.pairs(df)
```

Look for **FSharp.Data** on GitHub

Theme #3

Data and Types at Multiple Scales

Data at Multiple Scales

From Everything to Individuals

```
data.AllEntites
```

Data Scripters need to work with different granularities of schematization

```
data.Automotive.``Automobile Models``
```

```
data.Automotive.``Automobile Models``.Individuals.``Porsche 911``
```

…Only a language that supports massively scalable metadata can operate at all these levels

Every stable entity can get a unique type

Theme #4

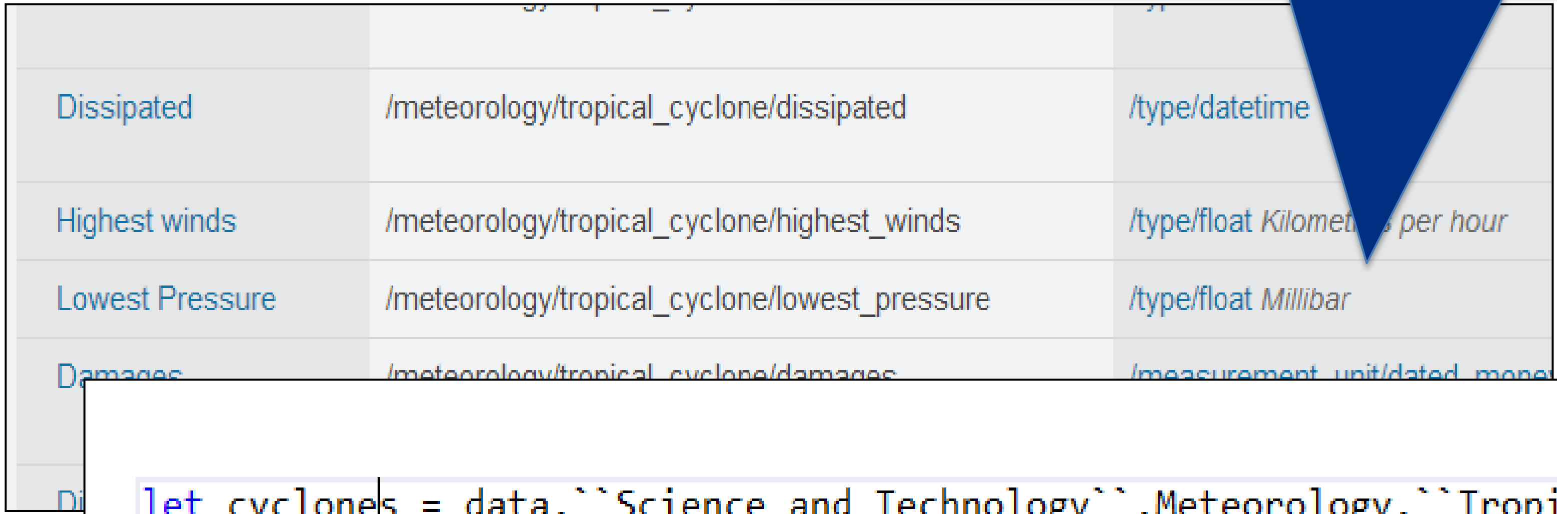
Programming Type Systems v. Information Space Metadata Synergy or Conflict?

Examples: Types, Schema, Constraints, Units of Measure, Security Information, Documentation, Definition Locations, Help , Provenance, Privacy, Ratings, Rankings, Search…

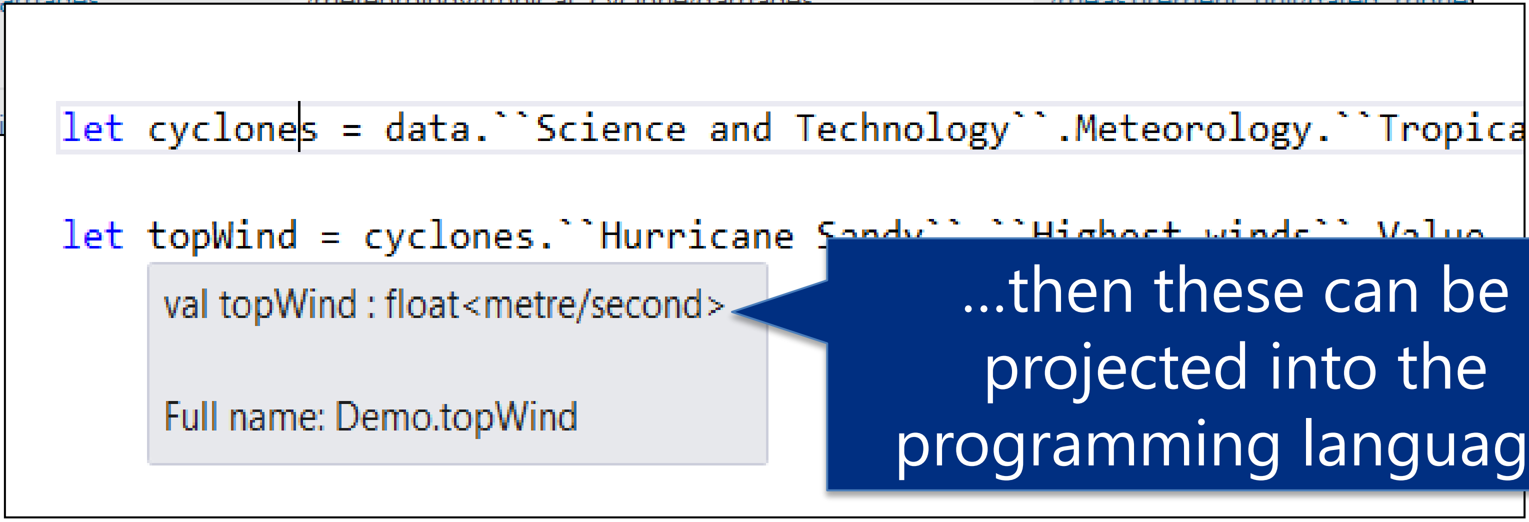
Providing Units of Measure

via F#'s Units of Measure

If the metadata contains units…



Dissipated	/meteorology/tropical_cyclone/dissipated	/type/datetime
Highest winds	/meteorology/tropical_cyclone/highest_winds	/type/float *Kilometres per hour*
Lowest Pressure	/meteorology/tropical_cyclone/lowest_pressure	/type/float *Millibar*
Damages	/meteorology/tropical_cyclone/damages	/measurement_unit/dated_money



```
let cyclones = data.``Science and Technology``.Meteorology.``Tropica

let topWind = cyclones.``Hurricane Sandy``.``Highest winds``.Value
```

```
val topWind : float<metre/second>

Full name: Demo.topWind
```

…then these can be projected into the programming language.

Theme #5

Schema Change

Hard problem, some progress…

- Many, many data sources are surprisingly stable
- Some data sources support “snapshot dates” (c.f. Datomic, Freebase)
- Data scripting has low exposure
- Support “invalidation signals” from providers at design-time
- Erasure makes compiled code much less fragile

Some Research Questions

Can incorporate schema change policies?

Can we provide all the data in the enterprise (e.g. SAP or Reuters)?

Can richer constraints, security, privacy and provenance annotations be provided?

Can temporal and probabilistic metadata be useful for typing or tooling?

Temporal types?
Probabilistic types?
Reactive types?

Can we model and verify provider components?

Can we provided types be mutually recursive?

Related Work

- Ur
- Gosu
- Scala Type Macros
- Template Haskell
- Scheme/Lisp Macros
- Idris
- …

Questions?

Learn more

tryfsharp.org

fsharp.org