

# Functional-first Programming in an Information-Rich World

*Dr Kenji Takeda*

*Microsoft Research Connections*



# Recurring Problems in Software

**Getting things done**

**Efficiency**

**Correctness**

**Complexity**

# What's the Need?

**Developers delivering correct, efficient software, on-time**

This is the set of problems that F# helps solve

## Observation #1

At the core of every functional-first language is this:

**simple, correct, robust code for complex problems**

## Observation #2

A highly interoperable language allows **rapid, non-intrusive deployment** and **integration** of components

Functional code is a part of a larger solution. Your code can be rapidly integrated and deployed.

Observation #2 cont.

**Interoperable languages remove entire phases from the software development process**

No R  $\rightarrow$  C#

No Mathematica  $\rightarrow$  C++

## Observation #3

Strongly-typed functional languages  
**maintain efficiency**

comparable to C# and Java, and sometimes C++



## Observation #4

Strongly-typed functional languages  
**help analytical programmers tackle  
more complex problems**

# How Functional-first Helps

**Simple, correct, robust code**

**Interoperability eliminates entire phases**

**Strong typing gives efficiency**

**Analytical developers empowered to solve complex problems**

What is F# and why  
should I care?

# F# is...

...a **practical, functional-first programming language** that allows you to write **simple code** to solve **complex problems.**

# F# and Open Source

F# 2.0 compiler+library open source drop

Apache 2.0 license

[www.tryfsharp.org](http://www.tryfsharp.org)

<http://blogs.msdn.com/dsyme>

Simple code,  
Strongly typed

# Simplicity: Functions as Values




F#

```
type Command = Command of (Rover -> unit)
```

```
let BrakeCommand =  
    Command(fun rover -> rover.Accelerate(-1.0))
```

```
let TurnLeftCommand =  
    Command(fun rover -> rover.Rotate(-5.0<degs>))
```



```
abstract class Command  
{  
    public virtual void Execute();  
}  
abstract class RoverCommand : Command  
{  
    protected Rover Rover { get; private set; }  
  
    public RoverCommand(MarsRover rover)  
    {  
        this.Rover = rover;  
    }  
}  
class BrakeCommand : RoverCommand  
{  
    public BrakeCommand(Rover rover)  
        : base(rover)  
    {  
    }  
    public override void Execute()  
    {  
        Rover.Rotate(-5.0);  
    }  
}  
class TurnLeftCommand : RoverCommand  
{  
    public TurnLeftCommand(Rover rover)  
        : base(rover)  
    {  
    }  
    public override void Execute()  
    {  
        Rover.Rotate(-5.0);  
    }  
}
```

# Simplicity: Functional Data

C#

```
let swap (x, y) = (y, x)
```

F#

```
let rotations (x, y, z) =  
  [ (x, y, z);  
    (z, x, y);  
    (y, z, x) ]
```

```
let reduce f (x, y, z) =  
  f x + f y + f z
```

```
Tuple<U,T> Swap<T,U>(Tuple<T,U> t)  
{  
    return new Tuple<U,T>(t.Item2, t.Item1)  
}
```

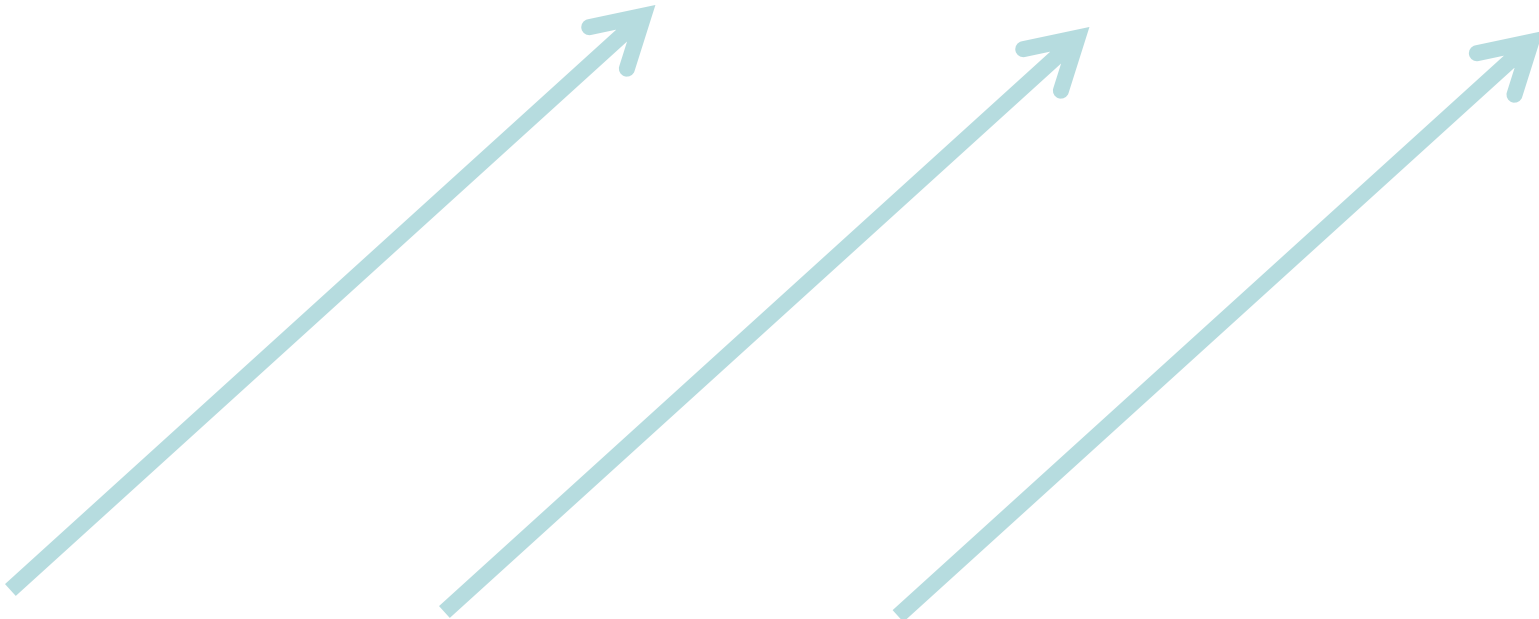
```
ReadOnlyCollection<Tuple<T,T,T>>  
Rotations<T>(Tuple<T,T,T> t)  
{  
    new ReadOnlyCollection<int>  
    (new Tuple<T,T,T>[]  
        { new Tuple<T,T,T>(t.Item1,t.Item2,t.Item3);  
          new Tuple<T,T,T>(t.Item3,t.Item1,t.Item2);  
          new Tuple<T,T,T>(t.Item2,t.Item3,t.Item1); });  
}
```

```
int Reduce<T>(Func<T,int> f,Tuple<T,T,T> t)  
{  
    return f(t.Item1) + f(t.Item2) + f (t.Item3);  
}
```



# The Big Trends

**THE WEB    MULTICORE    DATA**



# Parallel I/O

```
Async.Parallel [ httpAsync "www.google.com"  
                 httpAsync "www.bing.com"  
                 httpAsync "www.yahoo.com" ]
```

|> Async.RunSynchronously

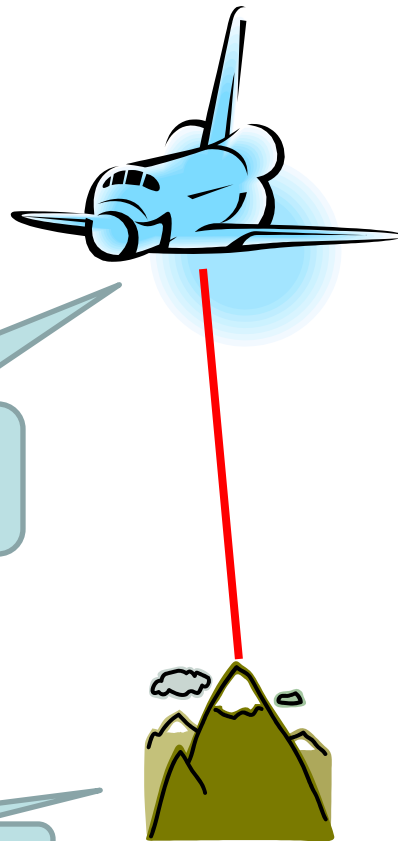
# Parallel CPU

```
Async.Parallel [ for i in 0 .. 200 -> computeTask i ]
```

```
|> Async.RunSynchronously
```

# Units of Measure

# 1985

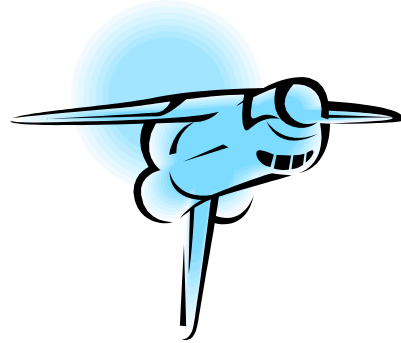


Mirror on underside  
of shuttle

Big mountain in Hawaii

## SDI experiment: The plan

# 1985



SDI experiment:  
The reality



### Attention All Units, Especially Miles and Feet!

Much to the surprise of Mission Control, the space shuttle Discovery flew upside-down over Maui on 19 June 1985 during an attempted test of a Star-Wars-type laser-beam missile defense experiment. The astronauts reported seeing the bright-blue low-power laser beam emanating from the top of Mona Kea, but the experiment failed because the shuttle's reflecting mirror was oriented upward! A statement issued by NASA said that the shuttle was to be repositioned so that the mirror was pointing (downward) at a spot *10,023 feet* above sea level on Mona Kea; that number was supplied to the crew in units of feet, and was correctly fed into the onboard guidance system -- which unfortunately was expecting units in nautical miles, not feet. Thus the mirror wound up being pointed (upward) to a spot *10,023 nautical miles* above sea level. The San Francisco Chronicle article noted that "the laser experiment was designed to see if a low-energy laser could be used to track a high-speed target about 200 miles above the earth. By its failure yesterday, NASA unwittingly proved what the Air Force already knew -- that the laser would work only on a 'cooperative target' -- and is not likely to be useful as a tracking device for enemy missiles." [This statement appeared in the S.F. Chronicle on 20 June, excerpted from the L.A. Times; the NY Times article on that date provided some controversy on the interpretation of the significance of the problem.] The experiment was then repeated successfully on 21 June (using nautical miles). The important point is not whether this experiment proves or disproves the viability of Star Wars, but rather that here is just one more example of an unanticipated problem in a human-computer interface that had not been detected prior to its first attempted actual use.

- [MAIN PAGE](#)
  - [WORLD](#)
  - [U.S.](#)
  - [LOCAL](#)
  - [POLITICS](#)
  - [WEATHER](#)
  - [BUSINESS](#)
  - [SPORTS](#)
  - [TECHNOLOGY](#)
  - [SPACE](#)**
  - [HEALTH](#)
  - [ENTERTAINMENT](#)
  - [BOOKS](#)
  - [TRAVEL](#)
  - [FOOD](#)
  - [ARTS & STYLE](#)
  - [NATURE](#)
  - [IN-DEPTH](#)
  - [ANALYSIS](#)
  - [myCNN](#)
- 
- [Headline News brief](#)
  - [news quiz](#)
  - [daily almanac](#)
- 
- MULTIMEDIA:**
  - [video](#)
  - [video archive](#)
  - [audio](#)
  - [multimedia showcase](#)
  - [more services](#)
- 
- E-MAIL:**
  - Subscribe to one of our news e-mail lists.
  - Enter your address:

## Metric mishap caused loss of NASA orbiter

September 30, 1999  
Web posted at: 4:21 p.m. EDT (2021 GMT)

### In this story:

- [Metric system used by NASA for many years](#)
- [Error points to nation's conversion lag](#)

### RELATED STORIES, SITES



NASA's Climate Orbiter was lost September 23, 1999

By Robin Lloyd  
CNN Interactive Senior Writer

(CNN) -- NASA lost a \$125 million Mars orbiter because a Lockheed Martin engineering team used English units of measurement while the agency's team used the more conventional metric system for a key spacecraft operation, according to a review finding released Thursday.

The units mismatch prevented navigation information from transferring between the Mars Climate Orbiter spacecraft team in at Lockheed Martin in Denver and the flight team at NASA's Jet Propulsion Laboratory in Pasadena, California.



# Units of Measure

```
let EarthMass = 5.9736e24<kg>
```

```
// Average between pole and equator radii
```

```
let EarthRadius = 6371.0e3<m>
```

```
// Gravitational acceleration on surface of Earth
```

```
let g = PhysicalConstants.G * EarthMass / (EarthRadius * EarthRadius)
```

```
let EarthMass = 5.9736e24<Ma
let EarthRadius = 6371.0e3<Ma
let g = Math.PhysicalConstant
let
  val g : float<m/s ^ 2>
```

```
///
```

# Formalizing an Extensional Semantics for Units of Measure

Andrew J. Kennedy  
Microsoft Research Cambridge  
akenn@microsoft.com

Interested in units of measure?

Kennedy, WMM 2008  
search for “kennedy units”

Bugs caused by units-of-measure errors can have catastrophic consequences, the most famous of which was the loss in 1999 of NASA's Mars Climate Orbiter probe [8], caused by a confusion between newtons (the SI unit of force) and lbf (the Imperial unit).

Many researchers have proposed preventing such errors at development time by type-checking [5, 6] or by static analysis [4, 2]. The former approach is exemplified by the Fortress programming language [1] and a recently-prototyped extension to I# [9]. Here, numeric types are parameterized by units, so `float<M/s^2>` represents an acceleration, and functions can be polymorphic in units, so `float<'u> -> float<'u^2>` is the type of `fun x->x*x`.

What is a *semantics* of units? An *intensional* approach would be to tag run-time values with their units, and then show that for type-correct programs the rules governing units are not broken at run-time. In contrast, we claim that the essence of unit correctness is the invariance of program behaviour under scaling: compare the well-known invariance of physical laws under scaling. Many consequences flow from this *extensional* interpretation: ‘theorems for free’, non-inhabitation of certain types, and most interestingly of all, type isomorphisms that mirror classical results from dimensional analysis. (An extensional study of *effects* is also fruitful [3].)

In previous work, these results were proved (by hand) for a calculus of explicitly-typed terms [7]. The aim now is to mechanize such results, in Coq, in a purely semantic framework based on parametric logical relations over underlying Coq values.

## Formalization and mechanization

Unit expressions are specified by  $\mu ::= u \mid \mathbf{1} \mid \mu \cdot \mu \mid \mu^{-1}$  where  $u$

and  $\models i \circ j = \text{id} : \sigma \rightarrow \sigma$ . It is straightforward to prove standard isomorphisms such as  $\tau \times \sigma \cong \sigma \times \tau$ . More interesting are unit-specific isomorphisms such as  $\forall u. \text{num } u \rightarrow \text{num } u \rightarrow \text{num } u \cong \text{num } \mathbf{1} \rightarrow \text{num } \mathbf{1}$ . For types of first-order functions, these are all instances of a direct analogue of the Pi Theorem from classical dimensional analysis. One purpose of the mechanization is to generalize this theorem to higher-order types.

## Experience

As is typical with mechanization, a number of techniques were attempted before settling on the cleanest approach. The general direction has been of increasing *abstraction*, preferring an abstract algebraic approach to a syntactic one. For example, rather than model unit expressions as abstract syntax trees, they are modelled as maps from unit variables to integers, so extensional equality is the right equality for units. Likewise, substitutions are modelled as homomorphisms over unit expressions.

De Bruijn encoding is used for unit variables and quantifiers. Unusually, environments are not lists, or finite maps, but are simply functions on all variables: when moving under a quantifier, we simply ‘shift’ the whole function making room for one more variable at the bottom.

Coq types mostly help, but sometimes hinder. A lemma stating  $\llbracket S(\tau) \rrbracket \psi = \llbracket \tau \rrbracket (\psi \circ S)$  for any substitution  $S$  is ill-typed, even though there is an easy proof of  $\llbracket \tau \rrbracket = \llbracket S(\tau) \rrbracket$ . The trick is to use this equality proof to construct coercions from  $\llbracket \tau \rrbracket$  to  $\llbracket S(\tau) \rrbracket$ .

The Setoid feature was used fruitfully to prove isomorphisms: once the definition of  $\cong$  is proved to be a congruence, complex isomorphisms can be built easily from primitive isomorphisms.

# Examples and Case Studies

# Example - power company

I have written **an application to balance the national power generation schedule** ... for an energy company.

**...the calculation engine was written in F#.**

The use of F# to **address the complexity at the heart of this application** clearly demonstrates a sweet spot for the language ... **algorithmic analysis of large data sets.**

Simon Cousins (Eon Powergen)

# Example power company

Time to Market

Efficiency

**Interoperation** ... Seamless. The C# programmer need never know.

**Parallelism** ... The functional purity ... makes it ripe for exploiting the inherent parallelism in processing vectors of data.

Correctness

**Units of measure** ... a huge time saver...it eradicates a whole class of errors...

Time to Market

**Code reduction** ... vectors and matrices...higher order functions eat these for breakfast with minimal fuss, minimal code. Beautiful.

Time to Market

**Exploratory programming** ...Working with F# Interactive allowed me to explore the solution space more effectively

Correctness

**Unit testing** ...a joy to test. There are no complex time-dependent interactions to screw things up....

Correctness

**Lack of bugs**... Functional programming can feel strange. .. once the type checker is satisfied that's often it, it works.

# Example - Biotech

...F# rocks - building algorithms for DNA sequencing and **it's like a drug**. 12-15 at Amyris use F#... A complete genome resequencing pipeline with interface, algs, reporting in ~5K lines and it has been incredibly reliable, fast and easy to maintain.. A suffix tree in 150 lines that can index 200,000 bases a second ;)

Efficiency

Correctness

Time to Market

**F# v. Python:** F# has been phenomenally useful. I could be writing a lot of this in Python otherwise and **F# is more robust, 20x - 100x faster to run and faster to develop.**

Correctness

**Units of measure:** I started labelling the coordinates as one or zero based and **immediately found a bug** where I'd casually mixed the two systems. Yay F#!

# Example - F# in Advertisement Ranking & Rating @ Microsoft

Time to Market

Around 95% of the code in these projects has been developed in F#.

- F# allowed for **rapid development of prototypes**, and thus also rapid verification or falsification of the underlying mathematical models.
- **Complex algorithms**, for example to compute Nash equilibria in game theory, **can be expressed succinctly**.
- **Units of measure** reduced the chance of errors dramatically: Prices, probabilities, derivatives, etc. can already be kept apart at compile time.

Taming Complexity

Correctness

# How Functional-first Helps

**Simple, correct, robust code**

**Interoperability improves time-to-market**

**Strong-typing gives efficiency**

**Analytical developers empowered to solve more complex problems**



Lesson: Combining with the right tools  
is key

# Examples

**A mathematical model**

- **F# + existing C++ components**

**A trading engine**

- **Oracle + F# (Server) + F#/C# (Silverlight)**

**A calculation engine with GPU execution**

- **F# + FCore Math**

**A scalable web service**

- **Data Services + F# + ASP.NET**

**A scalable big-data service**

- **F# + Hadoop + ServiceStack**

**A Web 2.0 startup**

- **F# (Server) + SQL Server + DataFeeds + ASP.NET + F# (WebSharper) + HTML5**

# Recent Developments in F# @ Microsoft

## F# 3.0

- queries, powerful data integration, better tooling, portable libraries

## F# + Azure

- for scalable service programming

## F# + Azure Hadoop

- for scalable big-data programming

## F# + Azure Cloud Numerics

- for scalable math programming

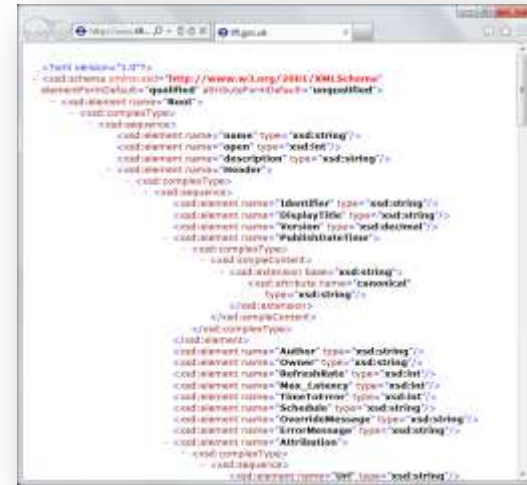


# Information-rich programming

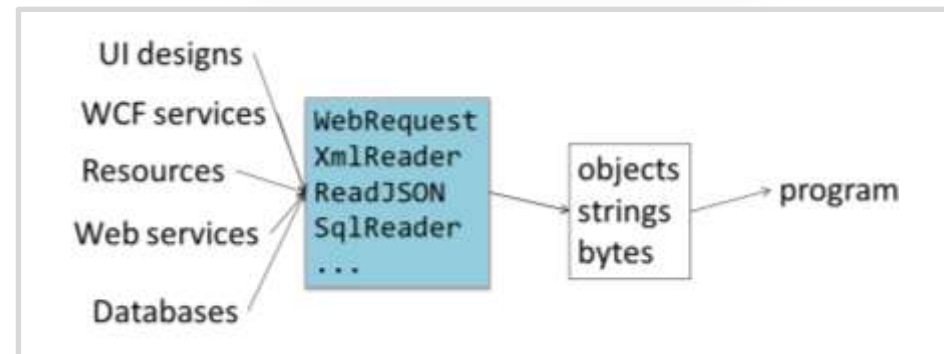
**“We live in an  
Information Society”**


# The developer's perspective

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



```
<?xml version="1.0"?>
<code xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://www.w3.org/2001/XMLSchema" xsi:type="xsd:string">
  <code element name="Root">
    <code complexType>
      <code sequence>
        <code element name="name" type="xsd:string"/>
        <code element name="open" type="xsd:bool"/>
        <code element name="description" type="xsd:string"/>
        <code element name="header">
          <code complexType>
            <code sequence>
              <code element name="Identifier" type="xsd:string"/>
              <code element name="DisplayTitle" type="xsd:string"/>
              <code element name="Version" type="xsd:decimal"/>
              <code element name="PublishStateBase">
                <code complexType>
                  <code simpleContent>
                    <code base="xsd:string"/>
                    <code attribute name="canonical" type="xsd:string"/>
                  </code>
                </code>
              <code element name="Canonical" type="xsd:string"/>
            </code>
          </code>
        </code>
      </code>
    </code>
  </code>
  <code element name="Author" type="xsd:string"/>
  <code element name="Owner" type="xsd:string"/>
  <code element name="Refreshable" type="xsd:bool"/>
  <code element name="Max_Latency" type="xsd:int"/>
  <code element name="TimeInterval" type="xsd:int"/>
  <code element name="Schedule" type="xsd:string"/>
  <code element name="OverrideMessage" type="xsd:string"/>
  <code element name="ErrorMessage" type="xsd:string"/>
  <code element name="Attributes">
    <code complexType>
      <code sequence>
        <code element name="URI" type="xsd:string"/>
      </code>
    </code>
  </code>
</code>
```



A person wearing a yellow long-sleeved shirt, blue pants, and a climbing harness is rappelling down a large, reddish-brown rock face. The person is positioned in the lower right quadrant of the frame, leaning forward with their arms extended. A rope is visible extending from the person towards the right edge of the image. The background consists of more rock formations under a clear blue sky.

A Big Problem

# New thinking required for languages

- Bringing information **into** the language
- *Solution: **Type Providers***
- Why F#?
  - Control of the language
  - Type inference
  - Strong tooling
  - Interoperability
  - LINQ
  - Open architecture





# A Type Provider is....

“A compile-time component that provides a computed space of types and methods on-demand ...”

“A compiler plug-in...”

“An adaptor between data/services and the .NET type system...”

**Note: F# still contains no data**

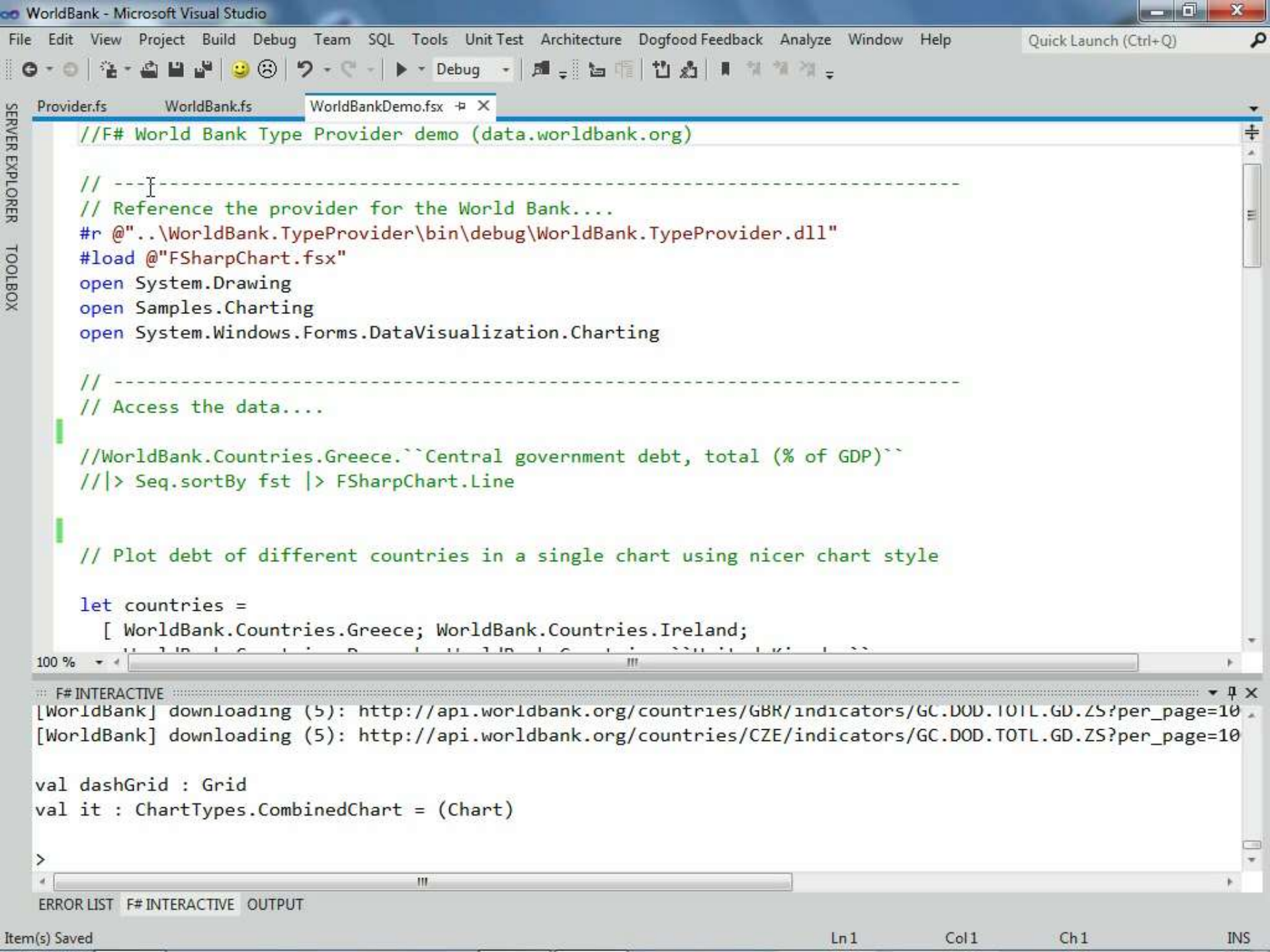
**Open architecture**

**You can write your own type provider**

# Intellisense for Data

The screenshot shows a web browser window displaying the World Bank Data website. The browser's address bar shows the URL <http://data.worldbank...>. The website header includes the World Bank logo and the tagline "Working for a World Free of Poverty". A navigation menu is visible with options: ABOUT, DATA (highlighted), RESEARCH, LEARNING, NEWS, PROJECTS & OPERATIONS, PUBLICATIONS, COUNTRIES, and TOPICS. Below the navigation is a red banner with the word "Data". Underneath, there are tabs for "By Country", "By Topic", "Indicators", "Data Catalog", "Microdata", "News", "About", "For Developers", and "Products". The main content area features a large image of wind turbines and a headline: "Open Climate Data Meeting on January 31st". The text below the headline reads: "Join us at the World Bank for a discussion about AppsForClimate, and using open data to address the challenges of climate change." To the left of the headline, there is a section titled "Open Data" with the text: "The Data Catalog provides download access to over 7,000 indicators from World Bank data sets." Below this, there is a search box labeled "Find an indicator" with the text "GNI per capita, Atlas method (current US\$)" and a "Go" button. A red button with the text "Learn more about data sets" is positioned below the search box. At the bottom left, there is a "BROWSE DATA" link, and at the bottom right, there is a "Feedback" button.

video



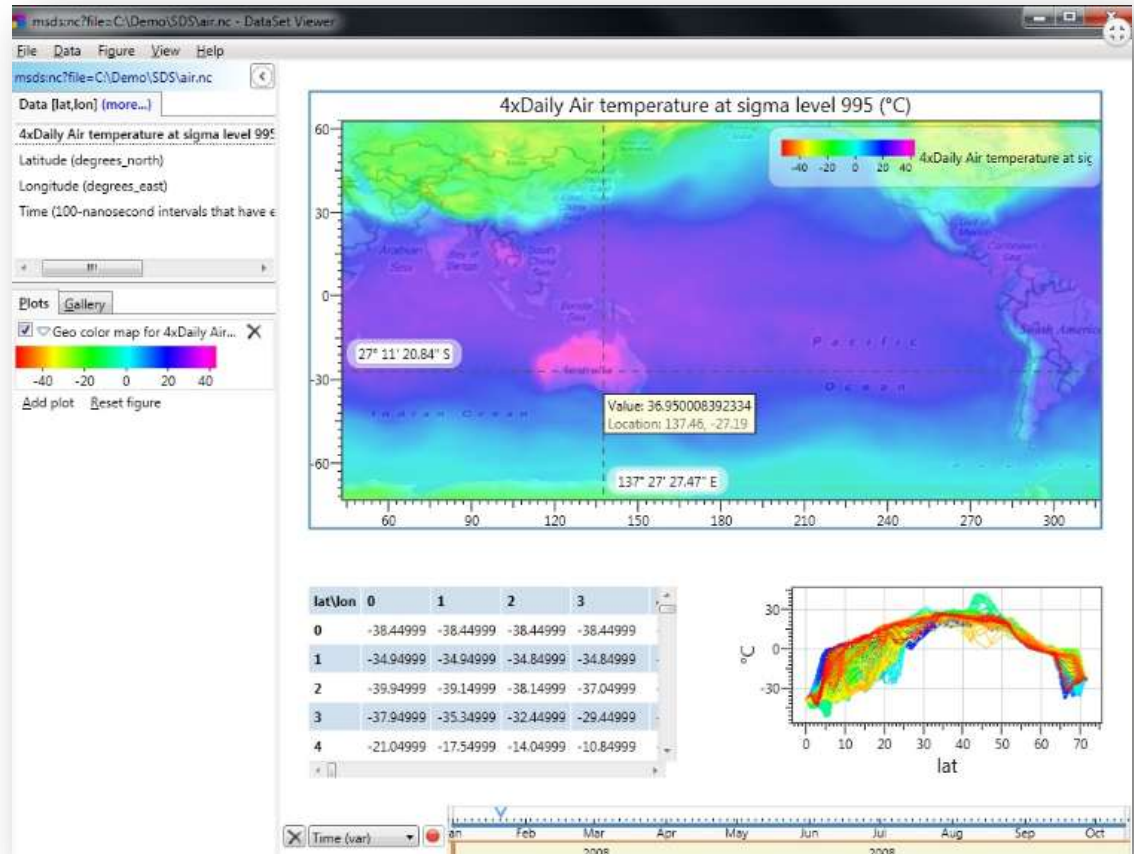
# Complex data



NCAR



Met Office



video





CEESDemoFINAL.fsx test.fsx

```
open System
#r "Microsoft.Research.Science.Data"
open Microsoft.Research.Science.Data
open namespace Microsoft.Research.Science.Data.Imperative

// referencing SDS type provider
#r @"C:\Users\kenjitak\Documents\Projects\F#\Demos\Data.SDS\Data.SDS\bin\Debug\Data.SDS.dll"

// starting DataSet Viewer connected to the 'view' DataSet
type viewType=Data.SDS.DataSet<"c:/Demo/template.csv?openMode=readOnly">

let view = viewType(DataSet.OpenSharedCopy("c:/Demo/template.csv"))
view.untyped().SpawnViewer("c:/Demo/Demo.dsvx")

// high res grid of benchmark monthly mean temperature for 1960-1990 from Climate Research Unit
type CRU=Data.SDS.DataSet<"C:/Demo/grid_10min_tmp.nc?openMode=readOnly">
let cru=CRU()

// select area of British isles and current month
```

100 %

F# INTERACTIVE

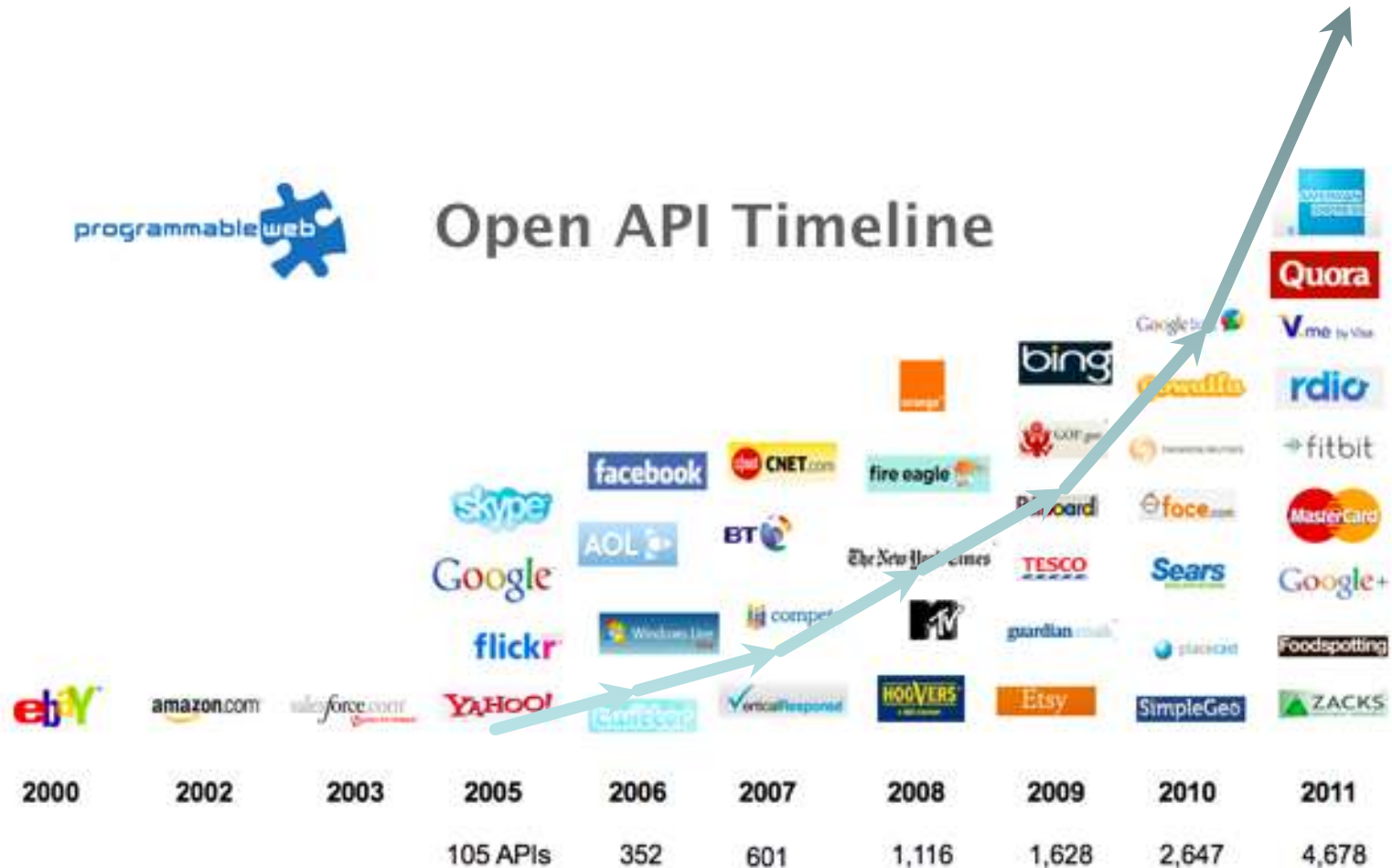
```
Microsoft (R) F# 3.0 Interactive build 11.0.50116.0
Copyright (c) Microsoft Corporation. All Rights Reserved.
```

```
For help type #help;;
```

```
>
```

F# INTERACTIVE OUTPUT

# Programming the web



# Type Providers: Applications

- ...web data
- ...data markets
- ...network management
- ...a spreadsheet
- ...web services
- ...CRM data
- ...social data
- ...SQL data
- ...XML data

**strongly  
typed**

**without  
explicit  
codegen**

**extensible,  
open**



# In Summary – Functional-First Languages

## Functional Programming

- **Functional-first languages deliver real value**
- **Rapid, correct development is central**
- **Parallelism a bonus**
- **F.P. as a recruitment strategy: languages are important, people even more so**

# In Summary – F#

**Improved time-to-market  
for analytical  
components**

**Ready for supported use  
in VS2010 + VS11**

**F#**

**Code correctness,  
efficiency and  
interoperation in the  
modern enterprise**

**A bright future ahead for  
web/data/cloud**

Learn more at  
**F# Tutorial Session: Wednesday@1700**

<http://fsharp.net>  
[www.tryfsharp.org](http://www.tryfsharp.org)



The Microsoft logo is displayed in a bold, italicized, black sans-serif font. The word "Microsoft" is followed by a registered trademark symbol (®). The logo is centered horizontally and positioned in the middle of the page. The background features a light blue gradient with several white, curved, glowing lines that create a sense of motion and depth, resembling a stylized globe or a futuristic interface element.

***Microsoft***<sup>®</sup>