Information Rich Programming with F# 3.0

Donna Malayeri, F# Program Manager
Microsoft

Agenda: F# 3.0

- F# 3.0 Information Rich Programming
- Type providers
- Query expressions
- Demos
- Type providers under the hood

F# Tagline

F# is a practical, functional-first language
that lets you write simple code
to solve complex problems

A Brief History of F#

- Microsoft Research Project (2002-2006)
  - Based on the functional language OCaml
  - Brainchild of Don Syme, key designer of .NET generics
- F# 2.0 made a product in Visual Studio 2010
  - Is popular for analytical computing
  - Industries such as banking, insurance, energy (and more!)
- F# 3.0 released in Visual Studio 2012

More about F#

- Full interop with .NET
  - Can call into .NET libraries and frameworks
  - Can be called from any .NET language
- Strongly typed, full type inference
  - Units of Measure feature reduces errors
  - Prevents mixing kilograms and ounces, dollars and euros (or your own custom units)

F# 3.0
Information Rich Programming

Two propositions

We live in an information society

Our languages are information sparse

A big problem for statically-typed languages

Challenges

- Impedance mismatch between types in language and types in data source
- Need to manually integrate codegen tools with build process, source control, etc.
- No elegant way to handle schema change
- Sometimes just have to up-cast to Object or parse strings

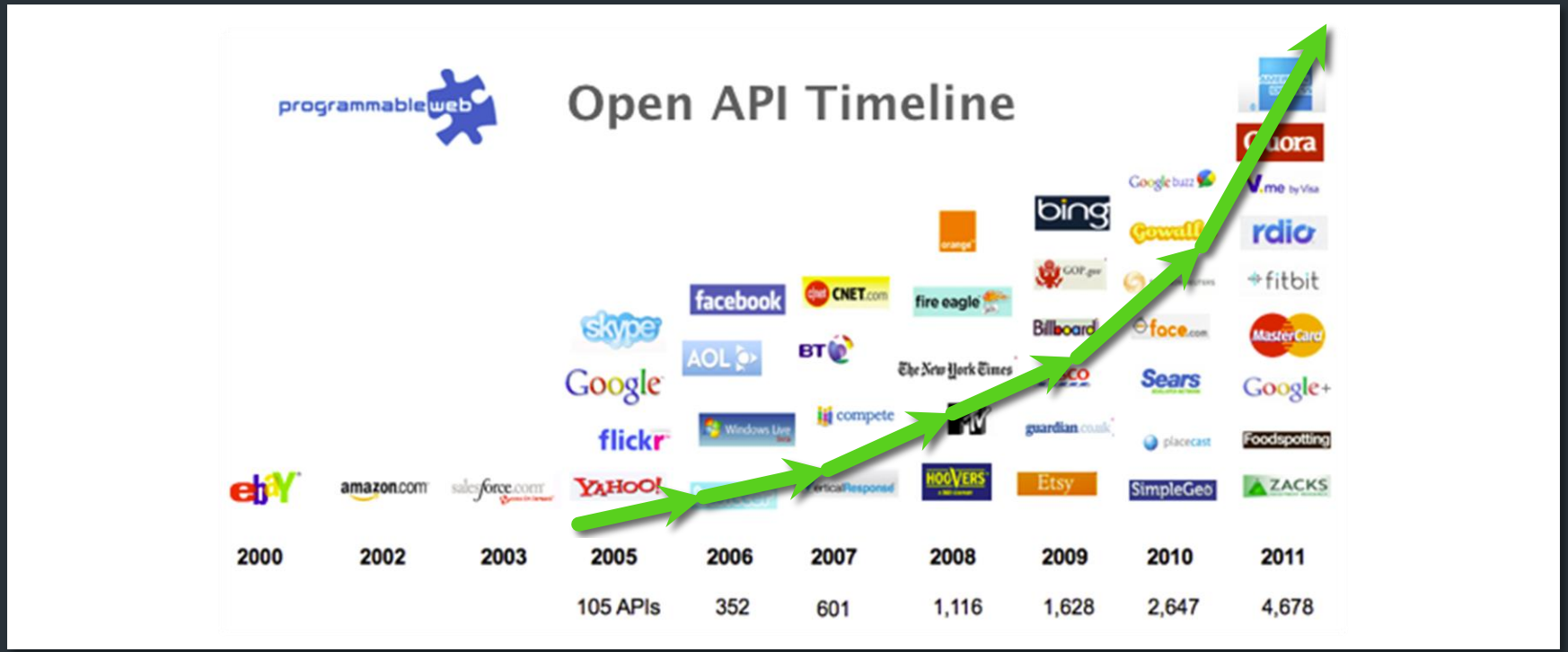
It doesn’t have to be this way!

- Data sources often have rich schemas and associated data definitions
- Static types should make your experience better, not worse!

Why This Matters

Programming the web

source: blog.programmableweb.com



How can we fix this?

Challenge some of our assumptions

- Compilers
- Tooling
- Language architecture

Challenge our notion of libraries

- Information spaces can be thought of as libraries that we use as part of the ambient programming environment
- E.g. .NET framework is part of the ambient environment of .NET languages

A type provider is…

- A design-time component that provides a computed space of types and methods
  - Intellisense for data
- A compiler/IDE extension
  - Extensible and open
- The static counterpart to dynamic languages

Task

Explore programming languages created after 1985

Freebase

page | discussion | view source | history

Freebase Documentation

Navigation

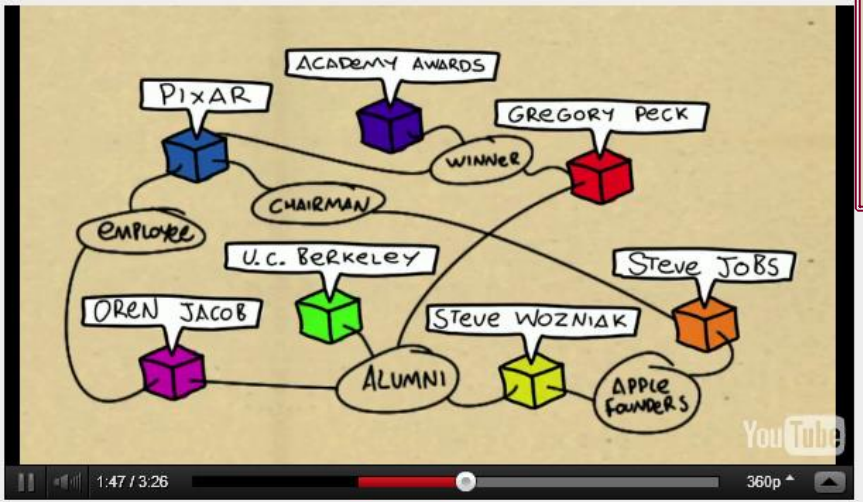
- Main page
- Recent changes
- Random page
- Help

Search

Go | Search

Toolbox

- What links here
- Related changes
- Special pages
- Printable version



Freebase is an open, Creative Commons licensed graph database with more than **22 million** entities.

An **entity** is a single person, place, or thing. Freebase connects entities together as a graph.

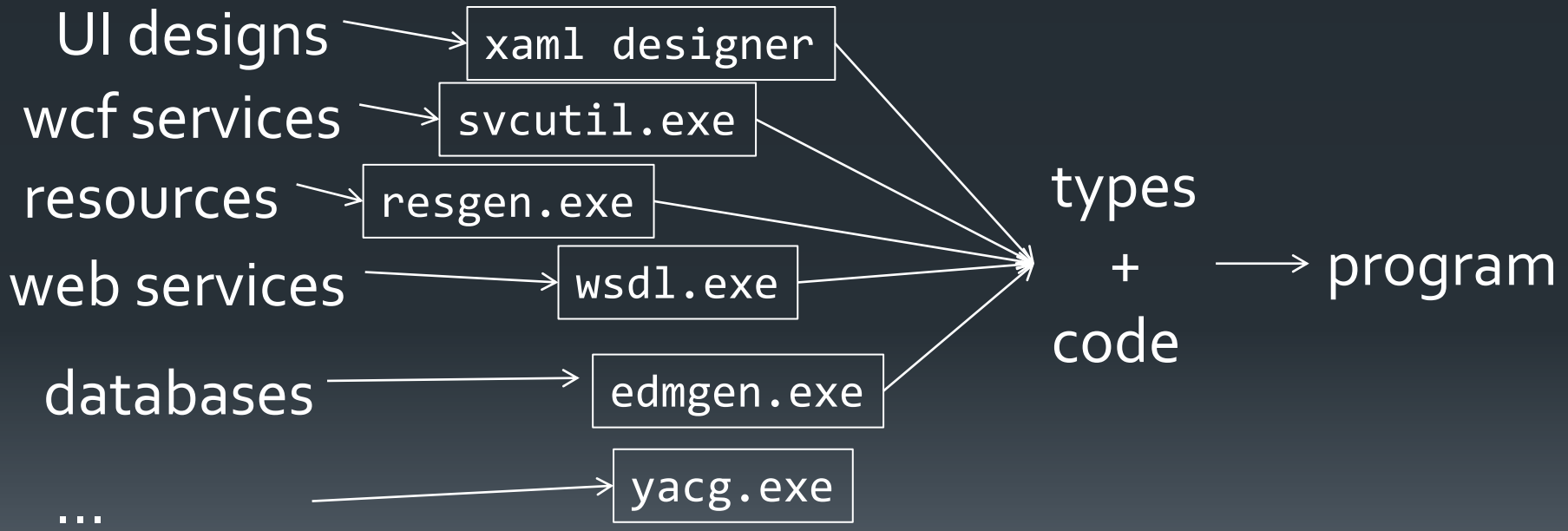
Ways to use Freebase:

- Use Freebase's **Ids** to uniquely identify entities anywhere on the web
- **Query** Freebase's data using MQL
- Build **applications** using our API or Acre, our hosted development platform

Freebase is also a **community** of thousands of data-lovers, working together to improve Freebase's data. Learn how to contribute, join our mailing list, or find out more on our community page.

Demo

How do we get the types today?



```fsharp
// Freebase.fsx
// Example of reading from freebase.com in F#
// by Jomo Fisher
#r "System.Runtime.Serialization"
#r "System.ServiceModel.Web"
#r "System.Web"
#r "System.Xml"

open System
open System.IO
open System.Net
open System.Text
open System.Web
open System.Security.Authentication
open System.Runtime.Serialization

[<DataContract>]
type Result<'TResult> = {
    [<field: DataMember(Name="code") >]
    Code:string
    [<field: DataMember(Name="result") >]
    Result:'TResult
    [<field: DataMember(Name="message") >]
    Message:string
    }

[<DataContract>]
type ChemicalElement = {
    [<field: DataMember(Name="name") >]
    Name:string
    [<field: DataMember(Name="boiling_point") >]
    BoilingPoint:string
    [<field: DataMember(Name="atomic_mass") >]
    AtomicMass:string
    }
```

```fsharp
let Query<'T>(query:string) : 'T =
    let query = query.Replace("'","\"")
    let queryUrl = sprintf
                   "http://api.freebase.com/api/service/mqlread?query=%s"
                   "{\"query\":"+query+"}"

    let request : HttpWebRequest = downcast WebRequest.Create(queryUrl)
    request.Method <- "GET"
    request.ContentType <- "application/x-www-form-urlencoded"

    let response = request.GetResponse()

    let result =
        try
            use reader = new StreamReader(response.GetResponseStream())
            reader.ReadToEnd();
        finally
            response.Close()

    let data = Encoding.Unicode.GetBytes(result);
    let stream = new MemoryStream()
    stream.Write(data, 0, data.Length);
    stream.Position <- 0L

    let ser = Json.DataContractJsonSerializer(typeof<Result<'T>>)
    let result = ser.ReadObject(stream) :?> Result<'T>
    if result.Code<>"/api/status/ok" then
        raise (InvalidOperationException(result.Message))
    else
        result.Result

let elements = Query<ChemicalElement
array>("[{'type':'/chemistry/chemical_element','name':null,'boiling_point':null
,'atomic_mass':null}]")

elements |> Array.iter(fun element->printfn "%A" element)
```

Freebase demo summary

- Can program against web-scale schematized data
  - (Codegen would never work here!)
- No waiting for codegen or compilations
- With typechecking
- Can detect schema change
- With great IDE tooling

Sure, but most schemas aren’t that big…

Codegen is not fun

Bing WSDL Services

Using the ServiceModel Metadata Utility Tool

Bing Maps SOAP Services are a set of web services built using Windows Communication Foundation (WCF) (http://msdn.microsoft.com/en-us/library/ms735119.aspx). The Bing Maps SOAP Services conform to WS-Basic Profile 1.1 (http://www.ws-i.org/Profiles/BasicProfile-1.1.html). It is highly recommended that you use the Service Model Metadata Utility Tool (**svcutil.exe**) or Visual Studio 2008 to generate the service proxy classes for the Bing Maps SOAP Services. Svcutil.exe can be downloaded from http://msdn.microsoft.com/en-us/library/aa347733.aspx or from the main menu in the Visual Studio

SOAP Services metadata URLs are listed in

ant to geocode an address and then

source file that contains service proxy

```
ervices http://dev.virtualearth.net

sses that have been generated. Add
```

Other Tools

If you are developing your application in Visual Studio and want to instead use the **wsdl.exe** command-line utility or the **Add Web Reference** menu item in the user interface, you need to set the corresponding *xxxSpecified* member to true for each member that is being set to enable the serialization of those members. More information about this can be found at http://blogs.msdn.com/eugeneos/archive/2007/02/05/solving-the-disappearing-data-issue-when-using-add-web-reference-or-wsdl-exe-with-wcf-services.aspx

Other Tools

If you are developing your application in Visual Studio and want to instead use the **wsdl.exe** command-line utility or the **Add Web Reference** menu item in the user interface, you need to set the corresponding *xxxSpecified* member to true for each member that is being set to enable the serialization of those members. More information about this can be found at http://blogs.msdn.com/eugeneos/archive/2007/02/05/solving-the-disappearing-data-issue-when-using-add-web-reference-or-wsdl-exe-with-wcf-services.aspx

It gets worse…

GeocodeServiceClient Class

msdn

Bing Services

Contains the methods used to make requests to the Geocode Service.

Note The name of this class and its constructor may be different depending on the tool you use to generate the client proxy classes.

C# VB

```
public class Geoc
```

Note The name of this class and its constructor may be different **depending on the tool you use to generate the client proxy classes**.

Constructor

Name	Description
GeocodeServiceClient	Initializes a new instance of a **GeocodeServiceClient** object.

Methods

Name	Description
Geocode	Finds a geographic location based on a request that may include the address, place, or entity type names to find.
ReverseGeocode	Finds geographic entities and addresses for a specified map location (known as

Demo: B-Movie Madness

Uniform access to a variety of data sources

Demo: Azure Marketplace

Demo summary

- Can easily program against multiple data sources using type providers
- Access web services, databases, etc, using a uniform interface
- F# works well for program logic

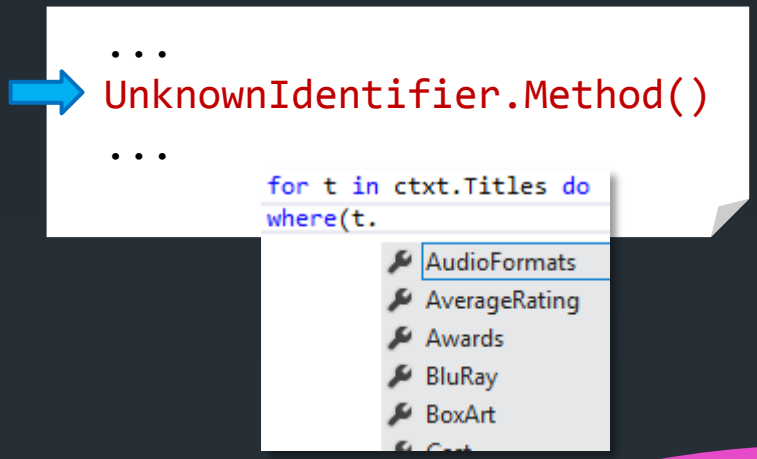
How do type providers work?

Under the hood

- A type provider is a standard .NET assembly
  - Extends a particular interface and includes special metadata attributes
  - Plugs into the compiler and typechecker
  - (We'll see the interface in a moment)

How it works

1. Typechecker encounters method application or type instantiation that is not in the environment
2. Typechecker queries all referenced type providers
3. Type provider returns a AST fragment and synthesized/real type definitions
4. Typechecker and compiler merge this into the rest of the AST



IDE Integration

Typechecker & Compiler

-r:Provider.dll

System.Type (synthetic)

+ how to compile method calls to synthetic types

Web Service
Database
Local file

Type provider interface (approx)

```
public interface ITypeProvider {

    Type GetType(string name);

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

    event System.EventHandler Invalidate;

    Type[] GetTypes();
}
```

Implementing a type provider

- Two alternatives
  - Erasure-based: inline the code, don't create types
  - Generated: inject IL into the assembly
- In principle, no unwanted references to external code/types

“Sort of” a plugin

- Extends the typechecker and compiler
- But is not a general compiler plugin
  - Can’t do arbitrary AST manipulation
  - Not inserting a phase in a pipeline
- A component that changes typechecking and IDE integration
  - Needs to work well in *both* batch and interactive mode
  - Get auto-completion “for free” since it's built into the typecheck process

Advantages

1. Can scale to huge schemas
   - e.g., a web database with millions of types
2. Strong, static types
   - Compile-time checking
   - Integrate with the IDE
3. Makes it easy to add new data protocols

Examples of type providers

- SQL
- Web services
- Structured files (CSV, XML,…)
- Regular expressions
- Facebook
- Data markets

Implementation: type provider with one type and one property

```
type SampleTypeProvider(config: TypeProviderConfig) as this =

    inherit TypeProviderForNamespaces()

    let oneType = ProvidedTypeDefinition("Samples.TypeSpace","OneType",Some typeof<obj>)
    let prop = ProvidedProperty( "Name",
                                 typeof<string>,
                                 GetterCode= (fun args -> <@@ “Hello" @@>))
    oneType.AddMember prop

    do this.AddNamespace("Samples.TypeSpace", [ oneType ])

[<assembly:TypeProviderAssembly>]
do()
```

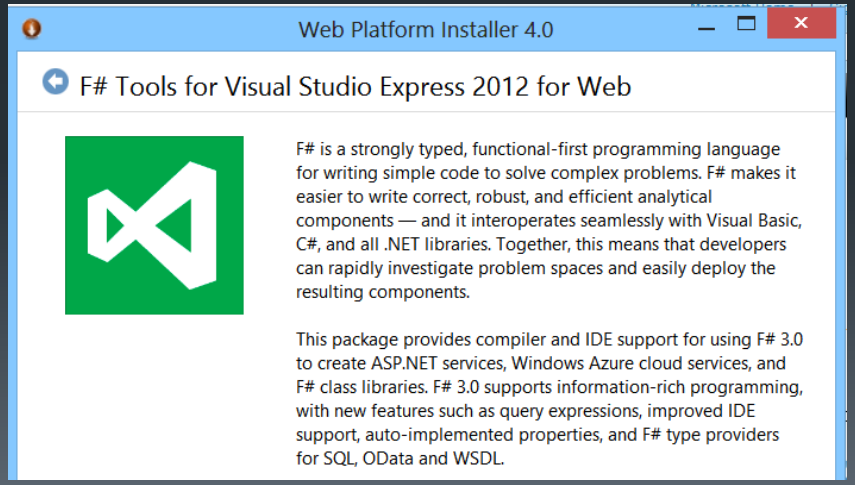
Built-in type providers in F# 3.0

- LINQ to SQL
- LINQ to Entities
- OData
- WSDL

F# Releases

Free Version of F#

- F# Tools for Visual Studio Express for Web
- Same F# 3.0 features as paid Visual Studio editions
- Go to http://fsharp.net



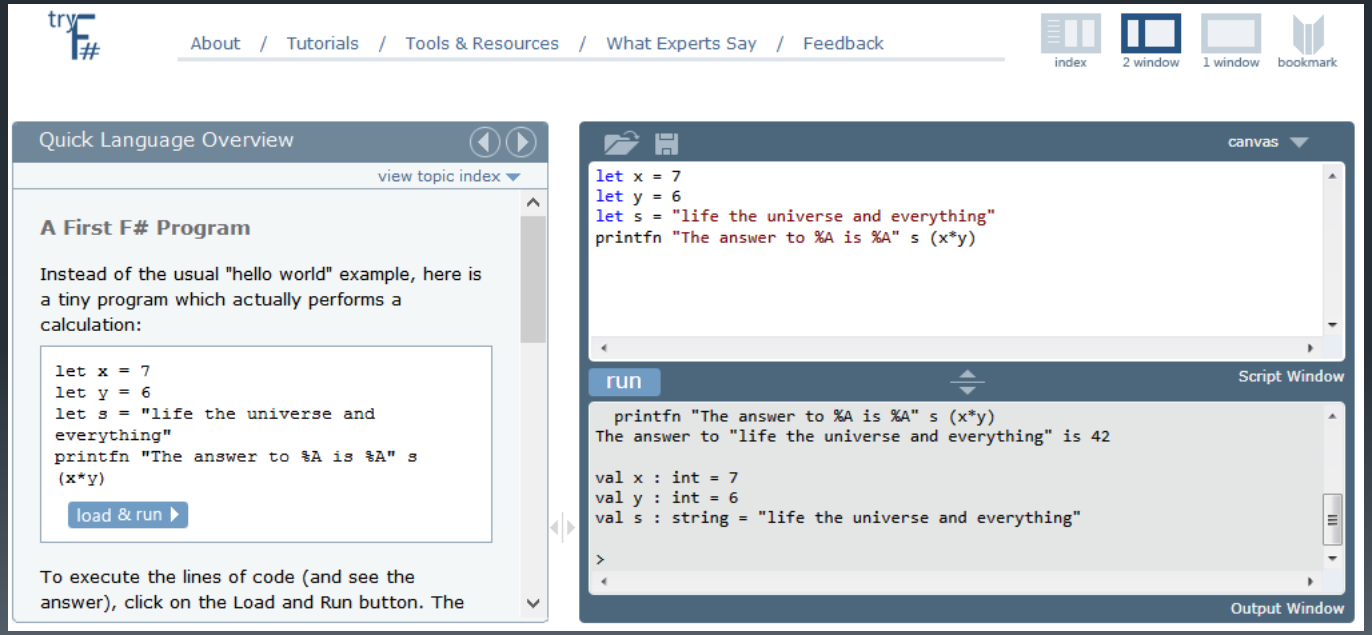
Open Source Releases

- F# 2.0 released under Apache 2.0 license
- F# 3.0 open source just released today!

- Both are available at fsharppowerpack.codeplex.com

Run F# In Your Browser

http://tryfsharp.org



Summary: F# 3.0

- Discover connected data and services—without ever leaving your editor
- Keeps the experience code-focused
- Provides a consistent and uniform programming experience
- Schema change integrates with IDE and build system
- Open and extensible